

**IN THE UNITED STATES DISTRICT COURT
FOR THE DISTRICT OF DELAWARE**

ORACLE CORPORATION and	)	
ORACLE U.S.A. INC.,	)	
	)	C.A. No. 06-414-SLR
Plaintiffs/Counterclaim Defendants,	)	
	)	JURY TRIAL DEMANDED
v.	)	
	)	PUBLIC VERSION
EPICREALM LICENSING, LP,	)	
	)	
Defendant/Counterclaim Plaintiff.	)	

**APPENDIX TO EPICREALM'S MEMORANDUM IN OPPOSITION TO ORACLE'S
MOTION FOR SUMMARY JUDGMENT OF INVALIDITY**

VOLUME II of II

EXHIBITS 27 to 42

OF COUNSEL:

Harry J. Roper
George S. Bosy
Aaron A. Barlow
Patrick L. Patras
David R. Bennett
Paul D. Margolis
Benjamin J. Bradford
Emily C. Johnson
JENNER & BLOCK
330 N. Wabash Avenue
Chicago, IL 60611-7603
Tel: (312) 923-8305

Richard L. Horwitz (#2246)
David E. Moore (#3983)
POTTER ANDERSON & CORROON LLP
Hercules Plaza, 6th Floor
1313 N. Market Street
Wilmington, DE 19899
Tel: (302) 984-6000
rhhorwitz@potteranderson.com
dmoore@potteranderson.com

*Attorneys for Defendant/Counterclaim
Plaintiff epicRealm Licensing, LP*

Dated: August 21, 2008
Public Version Dated: August 28, 2008
879525 / 31393 / Oracle

TABLE OF CONTENTS

<i>Exh.</i>	<i>Document</i>
1.	Excerpt from Deposition transcript of Paul C. Clark, July 11, 2008
2.	Excerpt from deposition transcript of Carol Colrain, April 15, 2008
3.	Excerpt from deposition transcript of Paul M. Dantzig, May 22, 2008
4.	Excerpt from deposition transcript of Dr. David Finkel, June 20, 2008
5.	Excerpt from deposition transcript of Steve Harris, April 18, 2008
6.	Excerpt from deposition transcript of Hal Hildebrand, Dec. 21, 2007
7.	Excerpt from deposition transcript of Ronald L. Howell, April 17, 2008
8.	Excerpt from deposition transcript of Keith Lowery, April 25, 2008
9.	Excerpt from deposition transcript of Katrina Montinola, Feb. 19, 2008
10.	Excerpt from deposition transcript of Mark Nelson, Feb. 22, 2008
11.	Excerpt from deposition transcript of Matthew Sarboraria, May 15, 2008
12.	Excerpt from deposition transcript of Michael Shamos, June 30, 2008
13.	Excerpt from deposition transcript of Douglas Surber, Feb. 28, 2008
14.	Atlanta Olympics WOMplex, March 1997, EPIC435843-435853
15.	USSN 08/794,269, Office Action, 8/23/2000, ORCL01623264-01623276
16.	Oracle10iAS Planning Update, ORCL01066383-01066413
17.	Oracle World, ORCL01050220-01050280
18.	Excerpt from Expert Report of Michael J. Wagner, 5/2/08
19.	Wagner Expert Report Volume 1 Tab 2, Schedules
20.	epicRealm's Second Amended Complaint, Jan. 27, 2006, <i>EpicRealm Licensing, LLC v. Autoflex Leasing, Inc. et al.</i> , (E. D. Tex.)
21.	Answer and Counterclaims of Safelite Group, Inc in Response to Plaintiff's Second Amended Complaint, February 8, 2006, <i>EpicRealm Licensing, LLC v. Autoflex Leasing, Inc. et al.</i> , (E. D. Tex.)
22.	Stipulation of Dismissal between epicRealm, LP and Safelite Group, Inc., June 26, 2006, <i>EpicRealm Licensing, LLC v. Autoflex Leasing, Inc. et al.</i> , (E. D. Tex.)

23. Order, Stipulation of Dismissal between epicRealm, LP and Safelite Group, Inc., June 29, 2006, *EpicRealm Licensing, LLC v. Autoflex Leasing, Inc. et al.*, (E. D. Tex.)
24. Settlement and License Agreement between Safelite Group, Inc and epicRealm Licensing, LP, June 7, 2006, EPIC435865-435872
25. US 5,761,673 Bookman, MAC003523-531
26. USSN 90/008,342, Reexam of US 5,894,554, Information Disclosure Citation, June 30, 2008
27. USSN 90/008,342, Reexam of US 5,894,554, Office Action, June 30, 2008
28. Excerpt from Report of Plaintiff's Expert Michael I Shamos
29. Excerpt from Oracle WebServer User's Guide, Release 1.0 A34986-2, ORCL00941713-00941960, *Ref.* 153 (also MAC003037-284)
30. Excerpt from Understanding SQL*Net Release 2.2, ORCL01806620-01806818
31. Functional Specification for Runtime Load Balancing, RDBMS, 10gR2, ORCL01060148-01060193
32. U.S. Patent Application 2005/0262183, Colrain et al., Nov. 24, 2005
33. InfoSpinner product information and correspondence, EPIC000133-000173
34. Letter re InfoSpinner products, EPIC014932-014936
35. Facsimile from R. West to S. Dettmer et al., EPIC014986-014991
36. Distributorship Agreement between Software AG of North America, Inc. and InfoSpinner, Inc., 2/26/96, EPIC000006-000034
37. Letter from R. West to J. Daly, 3/12/96, EPIC365224-365225
38. USSN 09/234,048, US 6,415,335, Paper 9, Response to Office Action, EPIC000489-501
39. Oracle WebServer 2.0 Technical Note March 1996, ORCL000761-000781, *Ref.* 605
40. Lagoze, et al., Dienst: Implementation Reference Manual, Cornell University Technical Report TR95-1514 (May 5, 1995). MAC 000270-000338
41. epicRealm Supplemental Response to Plaintiffs' Interrogatory Nos. 3-5, 9-15, 17-20, and 22-25, May 2, 2008
42. NTIS, Implementing Distributed Server Groups for the World Wide Web, 1/25/95, ORCL01818175-01818194

**IN THE UNITED STATES DISTRICT COURT
FOR THE DISTRICT OF DELAWARE**

CERTIFICATE OF SERVICE

I, David E. Moore, hereby certify that on August 28, 2008, the attached document was electronically filed with the Clerk of the Court using CM/ECF which will send notification to the registered attorney(s) of record that the document has been filed and is available for viewing and downloading.

I further certify that on August 28, 2008, the attached document was Electronically

Mailed to the following person(s):

Mary B. Graham
Morris, Nichols, Arsht & Tunnell LLP
1201 North Market Street
P.O. Box 1347
Wilmington, DE 19899
MGraham@MNAT.com

James G. Gilliland
Igor Shoiket
Townsend and Townsend and Crew LLP
Two Embarcadero Center
Eighth Floor
San Francisco, CA 94111-3834
OracleEpicrealm@townsend.com

Theodore T. Herhold
Robert J. Artuz
Eric M. Hutchins
Eric A. Mercer
Joseph A. Greco
Nitín Gupta
Townsend and Townsend and Crew LLP
379 Lytton Avenue
Palo Alto, CA 94301
OracleEpicrealm@townsend.com

Chad E. King
Townsend and Townsend and Crew LLP
1200 Seventeenth Street
Suite 2700
Denver, CO 80202
OracleEpicrealm@townsend.com

/s/ David E. Moore
Richard L. Horwitz
David E. Moore
Potter Anderson & Corroon LLP
Hercules Plaza – Sixth Floor
1313 North Market Street
P.O. Box 951
Wilmington, DE 19899-0951
(302) 984-6000
rhowitz@potteranderson.com
dmoore@potteranderson.com

Exhibit 27



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
90/008,342	11/27/2006	5894554		2076
8791	7590	06/30/2008	EXAMINER	
BLAKELY SOKOLOFF TAYLOR & ZAFMAN LLP 1279 OAKMEAD PARKWAY SUNNYVALE, CA 94085-4040			ART UNIT PAPER NUMBER	

DATE MAILED: 06/30/2008

Please find below and/or attached an Office communication concerning this application or proceeding.



UNITED STATES PATENT AND TRADEMARK OFFICE

Commissioner for Patents
United States Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450
www.uspto.gov

DO NOT USE IN PALM PRINTER

(THIRD PARTY REQUESTER'S CORRESPONDENCE ADDRESS)

DANIEL B. RAVICHER
PUBLIC PATENT FOUNDATION
1375 BROADWAY, SUITE 600
NEW YORK, NY 10015

MAILED

JUN 30 2008

CENTRAL REEXAMINATION UNIT

EX PARTE REEXAMINATION COMMUNICATION TRANSMITTAL FORM

REEXAMINATION CONTROL NO. 90/008,342.

PATENT NO. 5894554.

ART UNIT 3992.

Enclosed is a copy of the latest communication from the United States Patent and Trademark Office in the above identified *ex parte* reexamination proceeding (37 CFR 1.550(f)).

Where this copy is supplied after the reply by requester, 37 CFR 1.535, or the time for filing a reply has passed, no submission on behalf of the *ex parte* reexamination requester will be acknowledged or considered (37 CFR 1.550(g)).

Office Action in Ex Parte ReexaminationControl No.
90/008,34290 008 574
90 008 562Patent Under Reexamination
5894554Examiner
MARY STEELMANArt Unit
3992**-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --**

- a ☒ Responsive to the communication(s) filed on 13 May 2008. b ☐ This action is made FINAL.
 c ☒ A statement under 37 CFR 1.530 has not been received from the patent owner.

A shortened statutory period for response to this action is set to expire 2 month(s) from the mailing date of this letter. Failure to respond within the period for response will result in termination of the proceeding and issuance of an *ex parte* reexamination certificate in accordance with this action. 37 CFR 1.550(d). **EXTENSIONS OF TIME ARE GOVERNED BY 37 CFR 1.550(c).** If the period for response specified above is less than thirty (30) days, a response within the statutory minimum of thirty (30) days will be considered timely.

Part I THE FOLLOWING ATTACHMENT(S) ARE PART OF THIS ACTION:

1. ☐ Notice of References Cited by Examiner, PTO-892. 3. ☐ Interview Summary, PTO-474.
 2. ☒ Information Disclosure Statement, PTO/SB/08. 4. ☐ _____.

Part II SUMMARY OF ACTION

- 1a. ☒ Claims 1-11 are subject to reexamination.
 1b. ☐ Claims _____ are not subject to reexamination.
 2. ☐ Claims _____ have been canceled in the present reexamination proceeding.
 3. ☐ Claims _____ are patentable and/or confirmed.
 4. ☒ Claims 1-11 are rejected.
 5. ☐ Claims _____ are objected to.
 6. ☐ The drawings, filed on _____ are acceptable.
 7. ☐ The proposed drawing correction, filed on _____ has been (7a) ☐ approved (7b) ☐ disapproved.
 8. ☐ Acknowledgment is made of the priority claim under 35 U.S.C. § 119(a)-(d) or (f).
 a) ☐ All b) ☐ Some* c) ☐ None of the certified copies have
 1 ☐ been received.
 2 ☐ not been received.
 3 ☐ been filed in Application No. _____.
 4 ☐ been filed in reexamination Control No. _____.
 5 ☐ been received by the International Bureau in PCT application No. _____.
 * See the attached detailed Office action for a list of the certified copies not received.
 9. ☐ Since the proceeding appears to be in condition for issuance of an *ex parte* reexamination certificate except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte* Quayle, 1935 C.D. 11, 453 O.G. 213.
 10. ☐ Other: _____

cc: Requester (if third party requester)

Application/Control Number: 90/008,342 90/008,562
Art Unit: 3992 90/008,574

Page 2

DETAILED ACTION

I. Summary

This Office Action addresses the claims for which reexamination has been requested and a substantial new question of patentability has been determined to exist, specifically, claims 1-11 of U.S. 5,894,554 issued to Lowery et al.

II. Notice Regarding Certain Reexamination Issues

The patent owner is reminded of the continuing responsibility under 37 CFR 1.565(a), to apprise the Office of any litigation activity, or other prior or concurrent proceeding, involving Patent No. 5,894,554 throughout the course of this reexamination proceeding. See MPEP §§ 2207, 2282 and 2286.

Extensions of time under 37 CFR 1.136(a) will not be permitted in these proceedings because the provisions of 37 CFR 1.136 apply only to "a Patent Applicant" and not to parties in a reexamination proceeding. Additionally, 35 U.S.C. 305 requires that reexamination proceedings "will be conducted with special dispatch" (37 CFR 1.550(a)). Extension of time in *ex parte* reexamination proceedings are provided for in 37 CFR 1.550(c).

Information Disclosure Statement

Application/Control Number: 90/008,342 90/008,562

Page 3

Art Unit: 3992

90/008,574

Where patents, publications, and other such items of information are submitted by a party (patent owner or requester) in compliance with the requirements of the rules, the requisite degree of consideration to be given to such information will be normally limited by the degree to which the party filing the information citation has explained the content and relevance of the information.

The initials of the examiner placed adjacent to the citations on the form PTO /SB /08A and 08B or its equivalent, without an indication to the contrary in the record, do not signify that the information has been considered by the examiner any further than to the extent noted above.

Cited references lacking dates or not found in the file have been lined through.

III. References Cited

Rogers et al., USPN 5,701,451, file date Jun 7, 1995, issue date Dec 23, 1997

A World Wide Web browser makes requests to web servers on a network which receive and fulfill requests as an agent of the browser client, organizing distributed sub-agents sub-agents as distributed integration solution (DIS) servers on an Intranet network supporting the web server which also has access agent servers accessible over the Internet. DIS servers execute selected capsule objects which perform programmable functions upon a received command from a web server control program agent for retrieving, from a database gateway coupled to a plurality of database resources upon a single request made from a Hypertext document, requested

Application/Control Number: 90/008,342 90/008,562
Art Unit: 3992 90/008,574

Page 4

information from multiple data bases located at different types of databases
geographically dispersed, performing calculations, formatting, and other services prior
to reporting to the web browser or to other locations, in a selected format, as in a
display, fax, printer, and to customer installations or to TV video subscribers, with
account tracking. (Abstract)

Popp et al., USPN 6,249,291 B1, file date Sep 22, 1995, issue date Jun 19, 2001

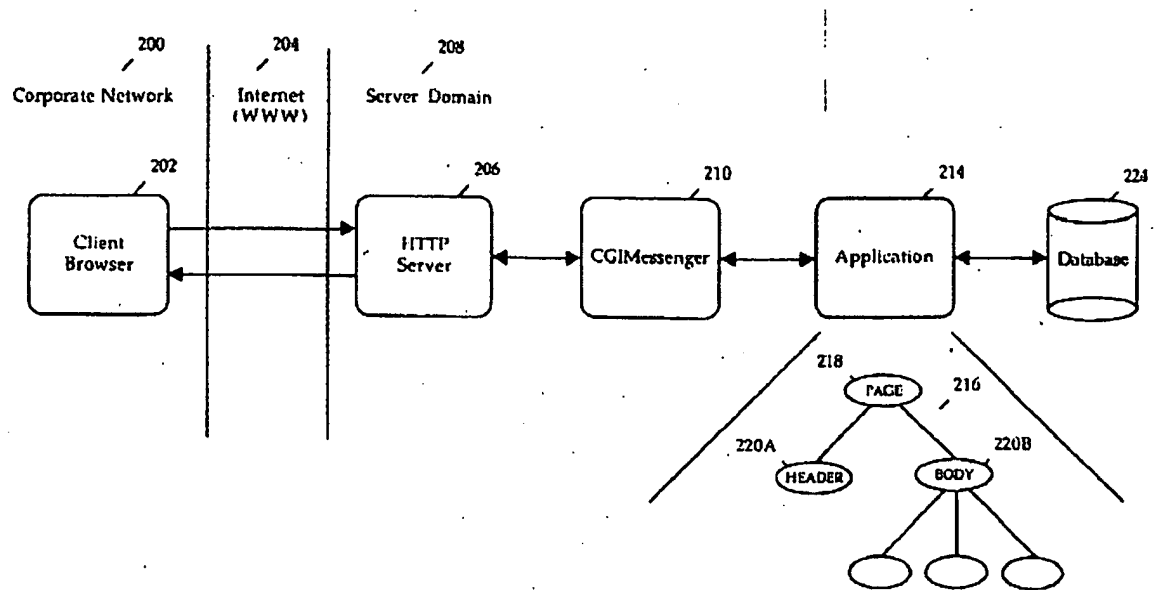
Popp discloses a system and method that "provides the ability to develop and manage Internet transactions." (Col. 3, 11.34-36.) In particular, a Web client can issue a request for a Web page to a Web server and "[s]ome or all of [the] Web page can be generated dynamically using input ... retrieved from an external data source (e.g., database ...)." (See, Abstract.) The Popp architecture is illustrated in Figure 2 of Popp (reproduced below). It comprises an HTTP server 206 (i.e., Web server), a CGIMessenger 210 and one or more applications 214 (i.e., page servers). (See generally, col. 6:32 - col. 7: 35.) Again, this architecture is substantially the same as the architecture described and claimed in the 554 patent; compare Figure 2 of Popp below to Figure 4 of the 544 patent.

Application/Control Number: 90/008,342 90/008,562

Page 5

Art Unit: 3992

90/008,574



As described in Popp, a request from a Client Browser 202 (i.e., Web browser) is received by the Web server 206. If it is determined from the URL of the request that the request calls for the generation of dynamic content, the "HTTP server 206 initiates CGIMessenger 210." (Col. 7:13 - 22.) That is, processing of the request by the HTTP server 206 is stopped, deflected or interrupted and the CGIMessenger 210 takes over. If the request identifies an application, such as application 214, "the CGIMessenger 210 transmits the client request ... from HTTP Server 206 to application 214." (Col. 7:25-28.) Thus, the CGIMessenger 210 functions as a "dispatcher" to dispatch the request to the application identified in the URL of the request. (Col. 7:5-7) (the "common gateway interface program ... is used to route requests to applications."). Inherent in the CGIMessenger 210 intercepting the request and forwarding it to application 214 (i.e., a page server), the Web server 206 is released from processing that request and can process other requests. Indeed, "CGIMessenger 210 can execute on the HTTP server

Application/Control Number: 90/008,342 90/008,562
Art Unit: 3992 90/008,574

Page 6

206 or another server connected to HTTP server 206, for example." (Col. 6: 57-58)

"Application 214 executes an interaction flow to satisfy the user request." Like the CGIMessenger 210, "[a]pplication 214 can execute on the same or different server as CGIMessenger 210 and/or HTTP Server 206." (Col. 7: 24-31.) The application 214 returns "a Web page (e.g., an HTML Web page) that is dynamically generated using complex queries (or other data retrieval mechanisms) to retrieve data and dynamically generate an HTML page using complex logic." (Col. 7: 44 - 58.) "Application 214 can access an external data source such as database 224. Database 224 can be resident on the same server as application 214 (Col. 7:31- 35.) In addition to the foregoing features, Popp also teaches that the system can be used to provide the notion of a "session" between a Web client and an application. Specifically, "[a] client's login password can be retained as state information at the session level, for example." (Col. 13: 33-35.) "Thereafter, the client does not have to enter the password to access an application during that session." Col. 13, 11.35-37. Thus, Popp teaches that the processing of requests may include the step of logging into (using a password, for example) the data source that an application accesses. Popp also teaches the use of HTML extension templates. The present invention provides an extension to HTML that is used on the HTTP server side. The HTML extension is filtered out before a Web page is sent to a client browser. It is used to interpret an HTML template and to render an HTML document before is transmitted to the client browser." (Col. 15: 45-50.) Specifically, Popp "uses an extension to the standard HTML known as the group extension. The group extension provides the ability to combine HTML elements or statements in a

Application/Control Number: 90/008,342 90/008,562

Page 7

Art Unit: 3992

90/008,574

single block." (Col. 15: 55-58.) "The group extension is identified by the <NSWTAG> and </NSWTAG> delimiters." (Col. 16: 31-32.) "[T]he group extension provides a link between an object that implements an HTML element and an object that implements a data item stored in a data source external to the WWW application." (Col. 15: 58 - 61.)

Installation and Planning Guide for Microsoft Internet Information Server, Version 1.0

("Installation Guide") (published Aug 1995) supplemented with ODBC Web Database Add-on for Microsoft Internet Server Beta Release Notes ("ODBC Notes") (Sep 27, 1995) (Microsoft References)

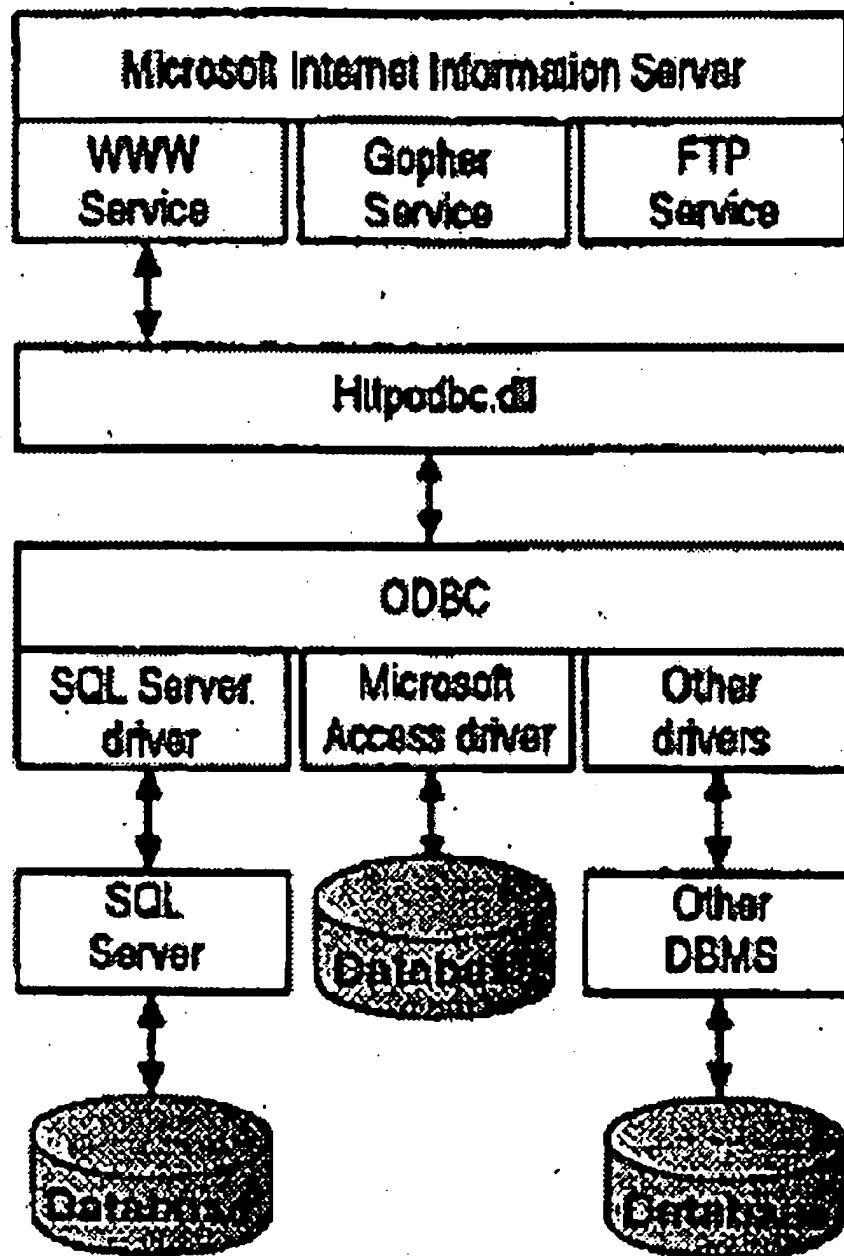
Both of the Microsoft References describe features of Microsoft Internet Information Server (or simply "Microsoft Internet Server" or "Microsoft Web Server" or "MS IIS"). The Microsoft References disclose that "[w]ith the Web Server and ODBC, [one] can create Web pages with information contained in a database," thus enabling generation of dynamic Web pages. (ODBC Notes, p. 1.) As shown in the following figure from the Installation Guide, Microsoft Internet Information Server with ODBC Add-on consists of multiple components: the Microsoft Internet Information Server with WWW Service; Httpodbc.dll (also called Database Connector), ODBC with SQL Server driver; SQL Server; and multiple databases. This architecture is virtually identical to the architecture for offloading generation of dynamic web pages disclosed and claimed in the '554 patent. Indeed, compare the figure below to Figure 4 of the 554 patent reproduced above.

Application/Control Number: 90/008,342 90/008,562

Page 8

Art Unit: 3992

90/008,574



As described in the Microsoft References, "Web browsers... submit requests to the Internet server by using HyperText Transport Protocol (HTTP). The Internet Server responds: with a document formatted in HyperText Markup Language (HTML). Access to databases is accomplished through a component of the Internet Server called Internet

Application/Control Number: 90/008,342 90/008,562
Art Unit: 3992 90/008,574

Page 9

Database Connector, one of family of Internet Server Extensions. The Database Connector is a Dynamic-Link Library (DLL) file, Httpodbc.dll, that uses ODBC to access databases. Httpodbc.dll uses two types of files to control how the database is accessed and how the output Web page is constructed. These files are Web Database Gateway (WDG) files and HTML extension (HTX) files. The WDG files contain the necessary information to connect to the desired ODBC data source and execute the SQL statement. A WDG file also contains the name and location of the HTX file.

The HTX file is the template for the actual HTML document that will be returned to the Web Browser after the database information has been merged into it by Httpodbc.dll.

(ODBC Notes, p. 2.) "On the Internet Server, the entire process of using the Database Connector is performed in six steps... Step 1" The URL is received by the Internet Server. The URL sent by the Web browser actually contains the name of the Database Connector, Httpodbc.dll, and the name of the WDG file to be used." (ODBC Notes, p. 4.)

"Step 2: The Internet Server loads Httpodbc.dll and provides it with the remaining information in the URL... Step 3: Httpodbc.dll reads the WDG file... Step 4: Httpodbc.dll connects to the ODBC data source, and executes the SQL statement contained within the WDG file. The connection is made to the ODBC data source by Httpodbc.dll, which in this example loads the ODBC driver for SQL server and connects to the server specified in the definition of the data source. Once the connection is made, the SQL Statement in the WDG file is sent to the SQL Server ODBC driver, which in turn sends it to SQL Server. Step 5: Httpodbc.dll fetches the data from the database, and merges it into the HTX file. Step 6: Httpodbc.dll sends the merged document back to the Internet Server,

Application/Control Number: 90/008,342 90/008,562
Art Unit: 3992 90/008,574

Page 10

which returns it to the client." (ODBC Notes, pp. 3-8.) As applied to the claims of the '554 patent, Microsoft Internet Information Server with WWW Server is a Web Server. It examines an incoming request to determine whether the request was made for a static or a dynamic page, and if the request is made for a dynamic page, it further determines which DLL file is specified (such as Httpodbc.dll) and then interrupts its processing of the request. When "the Database Connector Httpodbc.dll is referenced in the URL," the Web Server will then act as a dispatcher to dispatch the request to the Database Connector Httpodbc.dll, which handles generation of dynamic Web pages. (ODBC Notes, p. 3) Installation Guide at pp. 21-22 discloses that Microsoft Internet Information Server is capable of supporting multiple simultaneous connections. Typically, a single process (or thread) monitors TCP port 80 (the default port for Web requests) for requests. When a requests comes on port 80, the process that monitors the port necessarily has to spawn a different process. Otherwise, other requests that arrive to the Web Server on port 80 would not be processed until that single process finishes processing the previous request, thus rendering simultaneous processing of multiple connections impossible. Accordingly, to support multiple simultaneous connections releasing must necessarily occur. In Step 2 quoted above, "The Internet Server loads Httpodbc.dll and provides it with the remaining information in the URL," thus accomplishing the step of routing said request from said Web server to a page server. In Step 3 quoted above, "The Connection is made to the ODBC data source by Httpodbc.dll, which in this example loads the ODBC driver for SQL Server and connects to the server specified in the definition of the data source. Once the connection is made, the SQL Statement in the WDG file is sent to the SQL Server

Application/Control Number: 90/008,342 90/008,562

Page 11

Art Unit: 3992

90/008,574

ODBC driver, which in turn sends it to SQL Server." SQL Server serves as a data source for the dynamically generated web page that Httpodbc.dll creates. On pp. 21-22, Installation Guide discloses that Microsoft Internet Information Server is capable of supporting simultaneous connections, thus it is capable of processing requests while ODBC driver for SQL server simultaneously processes other requests. In steps 4, 5, and 6 quoted above, Httpodbc.dll retrieves data from one or more data sources and dynamically generates a Web page that includes the retrieved data. "After the SQL statement has been executed, Httpodbc.dll reads the HTX file specified in Sample.wdg (Sample.htx) and replaces every placeholder with the actual data returned by the query." (ODBC Notes, pp. 7-8.) While processing the request according to the six steps described above, Microsoft Internet Information Server with ODBC Add-on identifies a data source from which data is retrieved. Additionally, the Microsoft References disclose logging into data sources because a username is provided in a query. Microsoft Internet Information Server with ODBC maintains a connection cache to data sources. Specifically, Microsoft Internet Information Server enables logging of events into data sources. "Choose Log to SQL/ODBC Database to log activity information to any O.DBC data source. Set the Datasource name, Table, and specify the username and password to use when logging to the database." (Installation Guide, p. 59.) Maintaining a connection cache is inherent because it is impracticable to open and close a connection to a data source for every log entry. Microsoft Internet Information Server with ODBC maintains a page cache: "By default, Httpodbc.dll caches the Web page sent to the client. If a subsequent request is identical, the cached page will be returned without ever accessing the database." (ODBC

Application/Control Number: 90/008,342 90/008,562
Art Unit: 3992 90/008,574

Page 12

Notes, p. 12.) The Microsoft References also disclose HTML extension templates for configuring Web pages. "The Sample.htx file is an HTML document that contains placeholders for data returned from the database." (ODBC Notes, p. 7.) Data retrieved from data sources is inserted into those templates. "After the SQL statement has been executed, Httpodbc.dll reads the HTX file specified in Sample.wdg (Sample.htx) and replaces every placeholder with the actual data returned by the query." (ODBC Notes, p. 7.)

Oracle® Web Server User's Guide, Release 1.0 (1995) and Katrina Montinola, "The Oracle WebServer: A Technical Discussion," September 25, 1995 (collectively "the Oracle references")

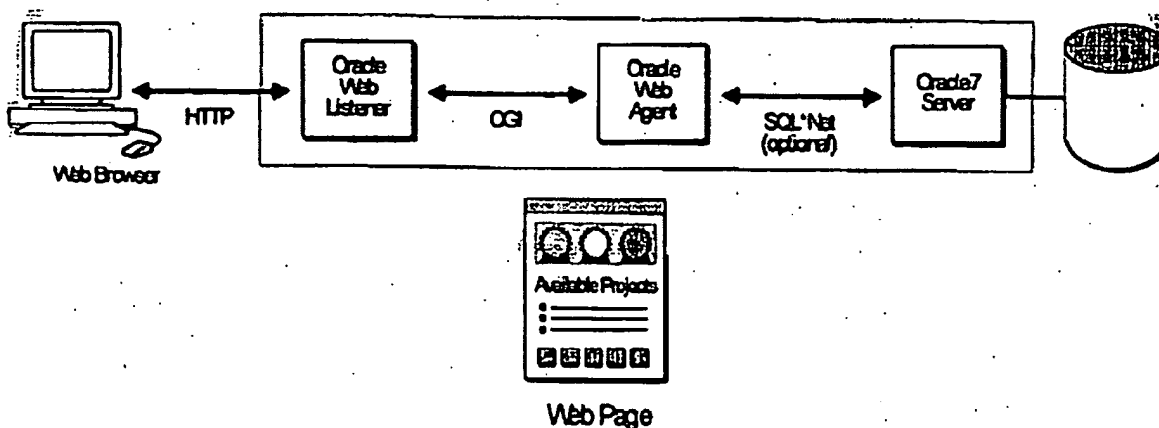
Both of the Oracle References describe Release 1.0 of Oracle's Web Server product. "The Oracle WebServer is an HTTP server with a tightly integrated Oracle 7 server that enables the creation of dynamic HTML documents from data stored in an Oracle database." (User's Guide, p. 1-2); see also, Montinola article, p. 2 ("the Oracle WebServer ... facilitates the creation of dynamic HTML documents ..."). As shown in the following figure from the User's Guide, the Oracle WebServer product consists of multiple components that closely correspond to the components of the partitioned architecture described and claimed in the 554 patent: the Oracle Web Listener, the Oracle Web Agent and the Oracle 7 Server. (User's Guide, p. 1-2; Montinola article, p. 1.) Indeed, a comparison of the figure below to Figure 4 of the 554 patent reproduced above reveals the similarities.

Application/Control Number: 90/008,342 90/008,562

Page 13

Art Unit: 3992

90/008,574



"The Oracle Web Listener is a commercial quality HTTP server that services document requests from any Web browser.... When the Oracle Web Listener receives a request from a client, it first determines whether that request is for a static document or a dynamic document." (User's Guide, p. 1-3; Montinola article, p. 3.) If the request is for a dynamic document, the Web Listener stops processing the request and invokes the Oracle Web Agent to process the request. Id. Thus, a request for a dynamic page is "intercepted" at the Web Listener and passed to the Web Agent. With reference to Fig. 4 of the 554 patent reproduced above, the Web Listener of the Oracle system corresponds to the Web server/interceptor blocks of the claimed system. "The Oracle Web Agent is a CGI program that ... logs into the [Oracle 7 Server] database and executes stored PL/SQL procedures that have been specified as part of the URL [of the received request]." (User's Guide, p. 5-2; Montinola article, p. 3.) "A PL/SQL procedure extracts data from the Oracle 7 database and generates an HTML document " (User's Guide; p. 5-1; Montinola article, p. 4.) "The HTML output generated by a PL/SQL procedure is sent from the Oracle 7 Server to the Oracle Web Agent." Id. "The output is then passed on to

Application/Control Number: 90/008,342 90/008,562
Art Unit: 3992 90/008,574

Page 14

the Oracle Web Listener." Id. "The Web Listener sends the generated HTML document back to the Web client." Id. Thus, in the context of the 554 patent the Web Agent serves as a "dispatcher" to dispatch a request for a dynamic Web page to the Oracle 7 Server which, in turn, functions as a "page server." "For maximum performance the Web Listener is designed to run as an asynchronous engine ... providing high performance under a heavy load." (User's Guide, p. 3-2.) Indeed, as mentioned above, when the Web Listener receives a request for a dynamic Web page, it "spawns the Web Agent." (User's Guide, p. 5-4; Montinola article, pp. 3-4.) Inherent in the "asynchronous" Web Listener "spawning" the Web Agent, the Web Listener is released or freed to process other incoming requests. To facilitate identification of the particular data source from which to retrieve data from the Oracle 7 database, the Web Agent may pass a QUERY_STRING environment variable to a PL/SQL procedure. (User's Guide, p. 5-4.) Specifically, "[i]f the PL/SQL procedure requires parameters, these would have been passed to the Web Agent through the QUERY_STRING environment variable." Id. "The Web Agent would then pass these on to the PL/SQL procedure." Id. For example, part of a query such as "person=Gretzky" may be passed to a PL/SQL procedure using the QUERY_STRING environment variable. (User's Guide, p. 5-9.) The PL/SQL procedure will then use the parameter at least in part to identify the data to retrieve.

The Web Agent must "log on" to the Oracle 7 Server in order to interact with it. (User's Guide, p. 5-4.) In order to log on, "the Web Agent requires certain information, such as which server to connect to and what username and password to use." Id. Specifically, the

Application/Control Number: 90/008,342 90/008,562

Page 15

Art Unit: 3992

90/008,574

Web Agent "[e]xtracts the database service name embedded in the URL [of the received request] and logs on to the database using the userid/password defined by the service." (Montinola article, p. 4.) The Oracle WebServer also employs page caching to improve performance. Specifically, "[t]he Oracle Web Listener allows a configurable set of commonly accessed files to be cached in memory. This provides very good performance when these files are accessed by a client." (User's Guide, p. 3-2.) The Oracle WebServer also provides custom HTML extension templates to facilitate dynamic Web page creation. Indeed, "one of the main goals of the Oracle Web Agent is to eliminate the PL/SQL programmer's need to be intimately familiar with World Wide Web Technology. To this end, the Oracle WebServer includes a Developer's Toolkit made up of several PL/SQL packages that minimize a programmer's need to know HTML syntax [B]y using the Toolkit he or she will not need to hard code the exact syntax of HTML tags into PL/SQL procedures [i.e., the procedures that extract data from the database and insert the data into an HTML document]." (User's Guide, p. 6-1); (see also Montinola article, p. 3) (the "WebServer Developer's Toolkit ... ease[s] the generation of HTML tags from within the user's PL/SQL code"). In greater detail, "[t]he Oracle WebServer Developer's Toolkit includes ... Hypertext Procedures (HTP)." (User's Guide, p. 6-1.) "A hypertext procedure generates a line in an HTML document that contains the HTML tag that corresponds to its name. For instance, the http.anchor procedure generates an anchor tag." Id. A Web developer uses these Hypertext Procedures in the PL/SQL code of a procedure designed to generate a dynamic Web page. There are a variety of Hypertext Procedures provided with the toolkit, including "Body Tags" that "are used in the main text of your HTML

Application/Control Number: 90/008,342 90/008,562
Art Unit: 3992 90/008,574

Page 16

page," "List Tags" that allow you to display information in lists, "Form Tags" that "are used to create and manipulate an HTML form," and "Table Tags" that "allow the user to insert tables and manipulate the size and columns of the table in a document." (User's Guide, pp. 6-1 to 6-31; also see, Montinola article, p. 3.) When a PL/SQL procedure retrieves data from the Oracle 7 database, it inserts the data into the Web page in accordance with the "template" defined by one or more of the Hypertext Procedures provided with the toolkit. For example, the "http.tableOpen" and "http.tableClose" Hypertext Procedures are used to insert retrieved data into an HTML table within a dynamic Web page. (User's Guide, pp. 6-29 to 6-30.) The above Hypertext Procedures constitute HTML extension templates that a Web developer can use to facilitate configuration of a dynamic Web page. In addition to the Hypertext Procedures provided as part of the toolkit, "[t]he design of the hypertext procedure ... package[] allows you to use customized extensions. Therefore, as the HTML standard changes, you can add new functionality similar to the hypertext procedure ... package[] to reflect those changes." (User's Guide, p. 6-37.)

John Gaffney, "Illustra's Web DataBlade Module, An Illustra Technical Paper," SIGMOD Record, Vol. 25, No. 1, March 1996 ("Illustra")

The Illustra reference discloses "a comprehensive toolset for creating Web-enabled database applications that dynamically retrieve and update Illustra database content."

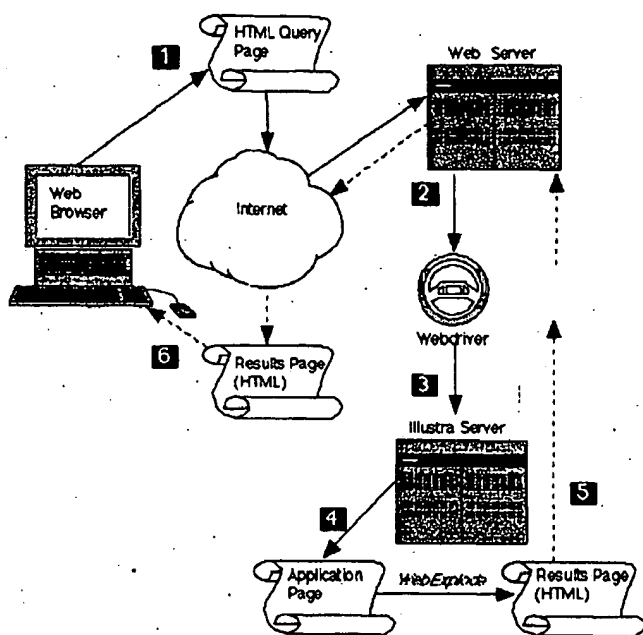
Application/Control Number: 90/008,342 90/008,562

Page 17

Art Unit: 3992

90/008,574

(Illustra, p. 105.) The basic components and operation of Illustra's Web architecture is illustrated in Figure 2, reproduced below.



Illustra's architecture has three tiers of which the Web Browser represents the first tier, which corresponds to the client of the 554 patent; the Web Server is the second tier, which corresponds directly to the Web server of the 554 patent; and Illustra Server represents the third tier, corresponding to the page server of the 554 patent. In the Illustra architecture the Web Server offloads the processor-intensive creation of dynamic Web pages to the Illustra Server. The handling of a request for a dynamically generated Web page in the Illustra architecture is as follows. "1. The Web browser queries the Web server with a form submission request or other query." (Id., p. 110.) The Web Server receives the request from the Web browser. "2. The Web server launches the client database application," such as Webdriver. (Id.) The Web Server must examine the request

Application/Control Number: 90/008,342 90/008,562
Art Unit: 3992 90/008,574

Page 18

to determine that launching of the Webdriver is required. Therefore the Web Server inherently intercepts the request. As shown in the figure above, after the Webdriver has been launched, the request is routed to it. Additionally, after routing the request to the Webdriver, the Web Server is no longer occupied with the processing of the request and it is therefore released from its processing. "3. Based on a defined set of parameters, the driver connects to the specified database as a specified user, and then selects a specified Application page..." (Id.) Not only the Webdriver identifies the database, but it also connects to it as a specified user, which requires logging in. "4. WebExplode parses the extracted pages, executes the embedded queries, and then formats the results. This is accomplished within the database server." (Id. p. 110.) The Application page is a template. It is stored in the database and is populated with data retrieved from database. (See Id., p. 108.) The embedded queries are SQL statements, which when executed retrieve data from the databases. As discussed above, WebExplode inserts the dynamically retrieved data into the Application Page. "5. Based on the results of WebExplode, an Application page is placed in the database server memory space and returned to Webdriver..." "6. Webdriver... renders the Application Page through the Web server to the client browser program." The Illustrations also disclose that "the requests to the database are all managed and processed through these already-established connections." (Id., p. 109.) Therefore, a connection cache is maintained.

Carl Lagoze, Erin Shaw, James R. Davis and Dean B. Krafft, "Dienst: Implementation Reference Manual," pp. 1-69 (May 5, 1995) ("Lagoze" and/or "Dienst")

Application/Control Number: 90/008,342 90/008,562
Art Unit: 3992 90/008,574

Page 19

Dienst discloses a protocol and server that provides distributed document libraries over the World Wide Web.

The Dienst Manual describes a system and method for managing requests from a WWW client via a WWW server for documents ("Web pages") stored on a page server (Dienst Server) or other data source such as a remote server. See generally Abstract, pages 6-11. The overall configuration is shown in Figure 2, at page 10.

Lagoze / Dienst includes a system and method for managing requests for documents ("Web Pages") stored on page server (Dienst server) or other server from WWW clients. The requester indicates that Dienst discloses the requests from the client being intercepted at the server by a CGI (Common Gateway Interface) stub as shown via figure 2 on page 10 of Dienst. The CGI stub dispatches the requests to a page server (Dienst server) which retrieves the requested dynamic content for delivery to the requesting client and which allows the WWW server to handle other incoming requests.

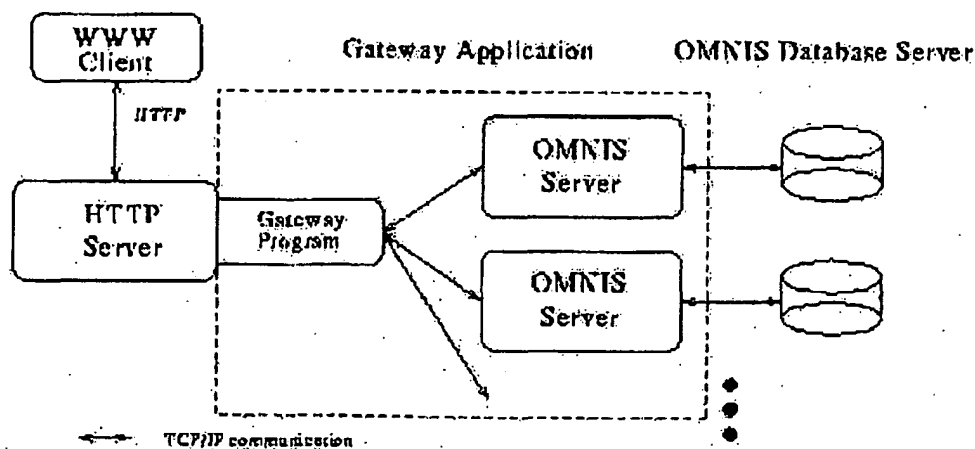
Alexander Clausnitzer, Pavel Vogel and Stephan Wiesener, "A WWW Interface to the OMNIS/Myriad Literature Retrieval Engine" (1995) ("Clausnitzer")

Clausnitzer describes a World-Wide-Web (Web) interface to a multimedia information search and retrieval system called "OMNIS/Myriad." The system is client/server-based

Application/Control Number: 90/008,342 90/008,562
 Art Unit: 3992 90/008,574

Page 20

and enables a user with a Web browser to search and retrieve documents and information about those documents from a database. As shown in Figure 5 of the reference (reproduced below), one embodiment of the system architecture includes an HTTP server (i.e., Web server) that receives HTTP requests containing OMNIS/Myriad search requests from a user's Web client (i.e., Web browser), a CGI "Gateway Program," and one or more OMNIS servers, each connected to a respective database of documents. (§ 3.2.) "The CGI program passes the WWW client's requests to one of the OMNIS servers which translate them into OMNIS queries for the database. The query results are sent back to the CGI program which transforms it into a WWW client readable format," i.e., a Web page. (Id.) Because each search request may be different and will produce different search results, each such request from a Web client constitutes a dynamic Web page generation request to the Web server.



Again, the architecture of the system described and illustrated in the Clausnitzer reference is virtually identical to the partitioned architecture disclosed and claimed in the 554 patent, as a comparison of the figure above to Figure 4 of the 554 patent once again

Application/Control Number: 90/008,342 90/008,562
Art Unit: 3992 90/008,574

Page 21

demonstrates. As described in the Clausnitzer reference, when the Web server recognizes from the URL in an HTTP request received from the Web client that the request is an OMNIS query (i.e., a dynamic Web page generation request), the Web server interrupts its processing of the request and passes the request to the CGI Gateway Program. (See §3.3 and description accompanying Figure 6.) Thus, the request is intercepted. The CGI program serves as a dispatcher, dispatching the request to one or more OMNIS servers. (§ 3.2 and Figure 5.)

Inherent in the CGI program accepting the request and forwarding it to an OMNIS server, the Web server process is released from processing that request and can process other requests. Indeed, Clausnitzer notes that "[w]ith fast following HTTP requests the CGI program can be started several times by the HTTP server to run parallel," thus confirming that the HTTP server continues to process other requests after starting the CGI program for a prior request. (§ 3.3.) "Each OMNIS server has a permanent connection to a single database," which serves as the "data source" from which the OMNIS server dynamically retrieves query results. (§ 3.2.) The OMNIS server to which a particular request is dispatched will process the query and return the results. The OMNIS server thus constitutes a "page server." "The query results are sent back to the CGI program which transforms it into a WWW client readable format," i.e., a Web page, which is then transmitted back to the client. (§ 3.2.) Because "[e]ach OMNIS server has a permanent connection to a single database," the connection that a given OMNIS server has to its data source is necessarily, maintained or cached. Also, OMNIS servers do require login and logout (see, § 3.2). Thus processing an OMNIS query necessarily requires at least

Application/Control Number: 90/008,342 90/008,562
Art Unit: 3992 90/008,574

Page 22

one initial login to the OMNIS server that processes the request. In addition to the foregoing features, Clausnitzer also teaches the use of custom HTML extension templates: A major point in the realization of the OMNIS-HTTP gateway was to make it easy to customize the pages and forms for different types of OMNIS databases. Since many databases should be available via WWW the gateway application has to provide easy methods for the administrator to add and change database parameters. For these purposes a special Mask Definition Language was developed. The masks consist of the fixed HTML fragments [i.e., custom HTML extension templates] which are filled by the gateway application and returned to the client as request results." (§ 3.3.)

Christian Derler, "The World-Wide Web Gateway to Hyper-G: Using a Connectionless Protocol to Access Session-Oriented Services," Institut für Informationsverarbeitung und Computergestützte neue Medien, pp. 1-104 (March 1995) ("Derler")

The Derler Thesis describes a system and method for managing requests from a WWW (World Wide Web) client via a WWW server (WWW Gateway) for documents or other information stored on a page server (Hyper-G Server) or other data source such as a document cache or remote server. See generally Abstract, Introduction at pages 1-2, and chapter 4 describing the WWW Gateway, particularly at pp. 48-55. The WWW Gateway includes software features described at pages 48-55, including Master, Passing, Slave, and Child processes, which intercept HTTP requests from the WWW client at the WWW Gateway, and dispatch such requests to the page server (Slave Processes and Hyper-G Server), while the WWW Gateway processes other requests

Application/Control Number: 90/008,342 90/008,562
Art Unit: 3992 90/008,574

Page 23

Bowman et al., "Harvest: A Scalable, Customizable Discovery and Access System" (March 12, 1995) ("Harvest")

Specifically, Harvest teaches "a system that provides an integrated set of customizable tools for gathering information from diverse repositories, building topic-specific content indexes, flexibly searching the indexes, widely replicating them, and caching objects as they are retrieved across the Internet. The system interoperates with WWW clients and with HTTP, FTP, Gopher, and NetNews information resources." (Harvest, p. 1.) Of particular relevance to claim 4 of the 554 patent, Harvest teaches the use of a connection cache. For example, Harvest explains that "[b]ecause [remote data retrieval] poses so much more load on provider-site servers, we implemented a connection caching mechanism, which attempts to keep connections open across individual object retrievals." (Id., at p. 10.) Harvest also teaches a page cache as found in claim 6 of the 554 patent. Specifically, Harvest explains that "[i]n addition to caching Gopher, HTTP, FTP, and NetNews objects, we maintain a cache of recent DNS name-to-address mappings to optimize common-case cache behavior." (Id., at p. 19.)

Antchev et al., "A WWW Learning Environment for Mathematics" (December 12, 1995) ("Antchev")

Application/Control Number: 90/008,342 90/008,562
Art Unit: 3992 90/008,574

Page 24

Antchev teaches a system for providing mathematic learning experiences over the Internet. (Antchev, p. 1.) To facilitate this, an "HTTP-server spawns [a] handler which is referred to by the URL of an incoming information request. The handler being invoked should return the appropriate HTML or other type of document or it should generate such a document on the fly." (Id. at p. 4.) Antchev, like Harvest, also teaches the use of a connection cache. For example, Antchev explains that because for "most of the exercises there might be several requests which form a session," a context tree data structure is used to store state information across an entire user session. Thus, Antchev maintains a "connection cache" across the session.

Ashley Beitz, Renato Iannella, Andreas Vogel, Zhonghua Yang, "Integrating WWW and Middleware"(May 2, 1995) ("Beitz")

Specifically, Beitz teaches an "architecture which enables WWW clients to access and invoke middleware services. In the architecture, the clients interact with the WWW, which in turn interacts with a range of middleware systems which provides access to the distributed system services." (Beitz, p. 2.) The Beitz reference is particularly relevant for its teaching of "logging into" a data source, as recited in claim 5 of the 554 patent. For example, Beitz teaches "filling in the necessary parameters to invoke [a] selected RPC operation." (Id. at p. 5.)

Ari Luotonen, and Kevin Altis, "World-Wide Web Proxies" (April 1994) ("Luotonen")

Application/Control Number: 90/008,342 90/008,562
Art Unit: 3992 90/008,574

Page 25

Specifically, Luotonen teaches a proxy server that "waits for a request from inside the firewall, forwards the request to [a] remote server outside the firewall, reads the response and then sends it back to [a] client." (Luotonen, p. 1.) Of particular relevance to claim 6 of the 554 patent is Luotonen's teaching of the use of a page cache. For example, Luotonen explains that "[a] brand new feature is [page] caching performed by the proxy, resulting in shorter response times after the first document fetch." (Id.)

Richard Knudson, "Application Development with Microsoft's Internet Information Server"
(February 2, 1996) ("Knudson")

The Knudson reference describes Microsoft's "Internet Information Server" product that allows for the generation of dynamic Web pages. Knudson, p. 1. The reference is particularly relevant because it teaches the use of "custom HTML extension templates" as recited in claims 7 and 8 of the 554 patent. Specifically, pages 3-4 of Knudson show examples of custom HTML extension templates into which data retrieved by the Internet Information Server is inserted when generating a dynamic Web page.

IV. Background and Overview

USPN 5,894,554 Lowery et al.

Application/Control Number: 90/008,342 90/008,562
Art Unit: 3992 90/008,574

Page 26

The '554 Patent issued on April 13, 1999. The patent generally relates to handling Web page requests, managing Web sites, and dynamic Web page generation. (Abstract & Col. 2:15-23).

The patent distinguishes some Web pages as having a static nature that remains static until manually modified and other Web pages as being dynamic Web pages which contain content that is generated dynamically by retrieving the necessary requested data and generating the requested Web page dynamically. (Col. 1:38-55).

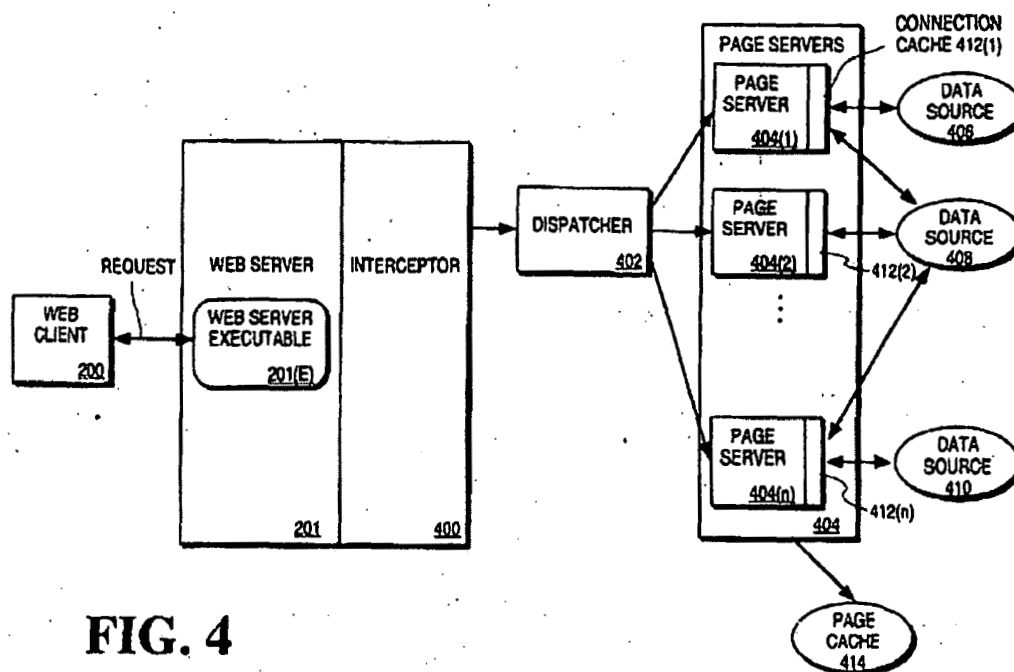
The patent is said to address a problem of managing numerous Web pages and Web page requests of Web clients at a Web site (Col. 2:2-12). With reference to Figure 4, reproduced below, the method is performed in a system including a Web server 201, separate page servers 404, and data sources 406, 408 and 410. The patent includes routing a Web request from the Web server to a Page server. Such routing is described as being performed by the dispatcher 402 (Col. 5 line 60 - col. 6 line 11). The routing frees the Web server to handle subsequent incoming requests. The Page server may then process the request, and the Web server is released to process other requests (Col. 2:20-35; Col. 4:54 - Col. 6:32). In this manner, dynamic Web pages may be generated by the Page servers.

Application/Control Number: 90/008,342 90/008,562

Page 27

Art Unit: 3992

90/008,574

**FIG. 4**

A review of the prosecution history reveals that Examiner initially rejected claims (CTNF 07/03/1997) based on USPN 5,532,838 to Barbari and "Beyond the Web: Excavating the Real World Via Mosaic" by Goldberg et al. Subsequent arguments persuaded Examiner to apply new art. A second non final office action applied USPN 5,404,527 to Irwin et al., to reject claim limitations. Arguments overcame claim rejections.

An Interview (12/29/1998) resulted in claim amendments that cancelled original dependent claim 2, independent claims 13 and 14, and dependent claim 15. Original claim 2 limitations were rolled up into remaining independent claims. Resulting independent claims (final claim

Application/Control Number: 90/008,342 90/008,562

Page 28

Art Unit: 3992 90/008,574

numbering: claim 1-method claim, claim 9 - system claim, and claim 11- machine readable medium version) recite similar limitations including:

...routing said request...wherein said routing step further includes the steps of intercepting said request...routing said request from said Web server/ second computer to a dispatcher, and dispatching said request to said page server...

The examiner did not indicate reasons for allowance during prosecution of the application which became the 5,894,554 patent to Lowery. It appears that the combination of limitations, as a whole, was not found in the prior art of record.

At the time of the Interview (12/29/1998) new related patent prior art was presented on a Form 892, citing USPN 5,761,673 to Bookman et al. and USPN 5,751,956 to Kirsch.

Bookman discloses a method and apparatus for generating dynamic Web pages. Specifically, the present invention claims a method and apparatus for generating dynamic Web pages on a Web server by invoking and executing predefined procedural packages stored in a database. The claimed invention receives an object request on the Web server and activates a Web agent on the Web server based on the object request. The Web agent invokes and executes the predefined procedural package to retrieve data from a data repository, and then formats the retrieved data as HTML output. (Abstract)

Application/Control Number: 90/008,342 90/008,562

Page 29

Art Unit: 3992

90/008,574

While Bookman appears to be relevant prior art, Bookman fails to fairly disclose "routing said request from said Web server to a page server", thereby freeing up the Web server to receive subsequent requests.

Kirsch discloses a Web server computer system which provides for server based controlled management over a client reference to a resource locator independently selected by a client computer system and referencing a server external Web server. The Web server system provides a client system with a predetermined URL reference to the Web server system encoded with predetermined redirection and accounting data including a reference to a second server system. On receipt by the first Web server system of the predetermined URL reference from the client system, the predetermined redirection and accounting data is decoded from the predetermined URL and processed by the Web server system to provide the client system with a redirection message including the reference to the second server system. The accounting data is processed by the Web server system and resulting data is selectively stored by the Web server system. (Abstract) Kirsch discloses a Web server that receives a client reference (request), tracks (accounting data) the client activity, and provides the client with a redirection message (col. 5: 37), "thereby minimizing the CPU and disk intensive load on the Web server computer system and the resulting latency (col. 5: 50-51) (i.e. client system is used to redirect to second server system)."

In contrast, the Lowry patent, '554, passes the client request from the Web server through a dispatcher, to a page server (second server) to minimize the CPU and disk intensive load on the

Application/Control Number: 90/008,342 90/008,562
Art Unit: 3992 90/008,574

Page 30

Web server. While Kirsch appears to be relevant prior art, Kirsch fails to fairly disclose a Web server that routes said request through a dispatcher, to a page server.

Both '554 and '956 patents attempt to reduce the workload of the Web server, but do it in different ways.

V. Claim Rejections – Relevant Statutes and Detailed Analysis

35 USC § 102

The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

A person shall be entitled to a patent unless –

(a) the invention was known or used by others in this country, or patented or described in a printed publication in this or a foreign country, before the invention thereof by the applicant for a patent.

Claims 1-11 are rejected under 35 U.S.C. 102(a) as being anticipated by Derler.

Claim 1.

[a] A computer-implemented method for managing a dynamic Web page generation request to a Web server, said computer-implemented method comprising the steps of:

The Derler Thesis discloses a computer-implemented method for managing dynamic Web page generation requests from a WWW client sent to a Web server (WWW Gateway). Introduction, pp. 1-2; § 3.3.4 at p. 37, § 4.1 at pp. 43-44; § 4.7.2 at pp. 94-96; Figs. 4.16-4.17 at p. 95. The WWW Gateway manages dynamic Web page generation requests from remote WWW clients

Application/Control Number: 90/008,342 90/008,562
Art Unit: 3992 90/008,574

Page 31

and thus constitutes a Web server. Operation of the WWW Gateway is a computer-implemented method, see generally Chapter 4.2 2. Derler's WWW clients use Web browsers to access information from the Hyper-G servers, therefore the Derler system is handling dynamic "Web page" generation requests. See Derler at p. 4 ("WWW clients, often called 'WWW browsers'"); p. 48; pp. 67-68 (explaining user interaction with Hyper-G systems with "WWW clients like Mosaic or Netscape", both of which are Web browsers).

[bl] routing said request from said Web server to a page server,

Derler: Web page requests are directed from the WWW Gateway (the "Web server") to the Hyper-G Server ("page server"). § 4.2.2 at pp. 48-49, § 4.2.3 at p. 50, 52, § 4.7.2 at p. 95, Fig. 4.17.

As described in Section 4.2.3, and accompanying Figures 4.1 - 4.7, a request from a client is initially received by a "Master Process" within the Web server. When the request is received, the Master Process "forks" generating a child process to process the request. "The master can close its connection to the client immediately after forking and is then ready to accept the next connection." If this is the first request from the Web client, the child process becomes a "Slave Process" (route directly to Slave Process / Page Server) which opens a connection to, and starts a session with, the Hyper-G server on behalf of the Web client. The Slave Process then sends a standard "entry page" back to the Web client, enabling the user to begin searching the Hyper-G server database. Subsequent search requests from the Web client are then received by the Master Process, which again forks to create a child process and to release the Master Process to process

Application/Control Number: 90/008,342 90/008,562
Art Unit: 3992 90/008,574

Page 32

other requests. For these subsequent requests, the child process becomes a "Passing Process" (route request from Web Server to Passing Process) that looks for the previously created Slave Process (page server) that holds the open connection to the Hyper-G server and then passes (dispatches) the client request to that Slave Process. The Slave Process will then transform the request into a Hyper-G request and send it to the Hyper-G server. The Hyper-G Server's response is transmitted back to the Slave Process. The Slave Process then transforms the response, as needed, and passes a Web page containing the results back to the Passing Process which relays it the Web client. In that respect, the Slave Process functions as a "page server," and the Hyper-G server functions as a data source. Thus, the system routes a client request from the Master Process of the Web server to a Slave Process that serves as a "page server" for handling the request.

[b2] said page server receiving said request and

As described in Derler, when the Master Process "forks" to pass a client request to a Slave or Passing Process, "the master can close its connection to the client immediately after forking and is then ready to accept the next connection." § 4.2.3. Derler: The client request is received by the slave process (page server) for processing. (See Id.)

[b3] releasing said Web server to process other requests,

Application/Control Number: 90/008,342 90/008,562
Art Unit: 3992 90/008,574

Page 33

Derler's WWW Gateway uses parallel processing of Web page requests wherein the WWW Gateway is released from one request to process other requests. See § 4.2.2 at pp. 48-49; § 4.2.3 at p. 49 (Master process of WWW Gateway), Figs. 4.1, 4.2, 4.3, 4.4 and 4.5 (pp. 52-55) illustrating process of releasing WWW Gateway to process other requests in parallel. Derler expressly recognizes that to avoid problems with throughput, "the process which accepts connections from the "outside" [i.e., from a Web client] is not burdened with the actual processing of the request, but instead is allowed to accept further requests while others are still being processed." (§ 4.2.2 -"Parallel Processing of Requests".)

[b4] wherein said routing step further includes the steps of intercepting said request at said Web server,

Derler: § 4.2.2 at pp. 48-49; § 4.2.3 at pp. 49-50; Figs. 4.16-4.17 and accompanying disclosure at pp. 94- 95; function of Master Process executing on WWW Gateway is to intercept Web page requests. Forked process is intercepted by a Passing Process.

[b5] routing said request from said Web server to a dispatcher, and

Derler: The Web Server routes the request to the Passing Process which in turn dispatches the request to the Slave Process (Page Server) that holds a connection to the Hyper-G server.

Passing Process identifies appropriate connection. § 4.2.2 at p. 48; § 4.2.3 at p. 50, Figs. 4.16-4.17 and accompanying disclosure at pp. 94-95 (depicting GET request path, including WWW

Application/Control Number: 90/008,342 90/008,562

Page 34

Art Unit: 3992

90/008,574

Gateway server to Hyper-G server); Web Server (Master) routes request to dispatcher (Passing Process is dispatcher/dispatches to identified Slave Process / Page Server), which looks for session keys to determine where to handle Web page request;

The Passing Process then dispatches the request to the Slave Process (page server) that holds an open connection to the Hyper-G server for requests from that particular client. As shown in Figures 4.1- 4.7, there may be many Slave Processes, and the Passing Process (dispatching the request to an identified slave process) must identify the appropriate one. The Passing Process thus functions to dispatch the request to the appropriate Slave Process.

[b6] dispatching said request to said page server;

Derler: Id.; The Passing Process then dispatches the request to the Slave Process (page server) that holds an open connection to the Hyper-G server for requests from that particular client. As shown in Figures 4.1- 4.7, there may be many Slave Processes, and the Passing Process (dispatching the request to an identified slave process) must identify the appropriate one. The Passing Process thus functions to dispatch the request to the appropriate Slave Process.

Figure 4.1, page server is Slave process which opens a connection to Hyper-G server (data source) and sends Hyper-G request to Hyper-G server. See Fig. 4.5 – 4.6, Slave A and Slave B process the request (page servers).

Application/Control Number: 90/008,342 90/008,562

Page 35

Art Unit: 3992

90/008,574

The Hyper-G server as stated in § 3.5 of Derler may perform a further requesting from a remote page server. The reference discloses a Hyper-G page server obtaining information from remote servers; the Hyper-G server fetches the information [from a remote server] and delivers it to the client, via a Slave Process (§ 4.4.2)

[c] processing said request, said processing being performed by said page server while said Web server concurrently processes said other requests; and

Derler: § 4.2.2 at pp. 48-49; § 4.2.3 at p. 49; pp. 52-54, processing of request by Hyper-G server; Figs. 4.3, 4.4, 4.5, such processing occurs concurrently with the WWW Gateway processing other requests. See also § 4.2.3 at p. 50 ("When it [a slave process on the WWW Gateway] has accepted a connection initiated by the passing process, it reads the request, processes it (usually by transforming it into a Hyper-G request for the server, receives the response from the Hyper-G server, transforms the response again if necessary, sends it to the passing service, closes the connection to the passing process.") Necessarily, since the client receives a response from the Hyper-G server, the Hyper-G server has processed the request. The slave process connects to Hyper-G server page server for processing of the request. See § 4.4.2, "The class WWWSlave contains all data and code necessary to process requests..."

As described in Derler, when the Master Process "forks" to pass a client request to a Slave or Passing Process, "the master can close its connection to the client immediately after forking and is then ready to accept the next connection (web server concurrently processes said other

Application/Control Number: 90/008,342 90/008,562

Page 36

Art Unit: 3992

90/008,574

requests)." § 4.2.3. Indeed, Derler expressly recognizes that to avoid problems with throughput, "the process which accepts connections from the "outside" [i.e., from a Web client] is not burdened with the actual processing of the request, but instead is allowed to accept further requests while others are still being processed." § 4.2.2 -"Parallel Processing of Requests".

[dl] dynamically generating a Web page in response to said request,

Derler: § 3.3.4 at p. 37; Fig. 4.10 at p. 71 (depicting sample of a dynamically-generated Web page in response to a search request). The Slave Process receives the results of the search from the Hyper- G server's database and generates a responsive Web page for the client, § 4.4.2 See, e.g., § 4.5.5 & Figure 4.13 (illustrating an exemplary Web page displaying the results of a search).

[d2] said Web page including data dynamically retrieved from one or more data sources.

Derler: Id.; data source for Web page can be data from a data cache, see § 3.3.5; see also p. 95, showing data retrieved from a remote data source in the form of a remote server.

Claim 2.

The computer-implemented method in claim 1 wherein said step of processing said request includes the step of identifying said one or more data sources from which to retrieve said data.

Application/Control Number: 90/008,342 90/008,562

Page 37

Art Unit: 3992

90/008,574

Derler: § 4.7.2, Fig. 4.17 (p. 95) (Hyper-G server identifies remote server from which to retrieve data); see also § 4.1 (integration of WWW databases into Hyper-G, inherently such databases would be identified so as to retrieve data from such WWW databases). See also § 3.2.3 ("Supporting bidirectional links can easily be done by keeping links and anchors in a separated database, called the link server. This link server is an object oriented database. It holds information about Hyper-G 'objects', which can be links, anchors, descriptions of documents, relations between such objects."); § 3.5 ("should information from a remote server be needed, the Hyper-G server which the client is connected to fetches the information and delivers it to the client.").

Claim 3.

The computer-implemented method in claim 2 wherein said step of dynamically generating said Web page includes the step of dynamically retrieving said data from said one or more data sources.

Derler: § 3.3.4 at p. 37; Fig. 4.10 at p. 71 (depicting sample dynamically-generated Web pages; data is retrieved dynamically from one or more sources); § 3.3.5 at p. 37 (dynamically retrieving data from document cache).

Claim 4.

Application/Control Number: 90/008,342 90/008,562
Art Unit: 3992 90/008,574

Page 38

The computer-implemented method in claim 3 wherein said step of processing said request includes the step of said page server maintaining a connection cache to said one or more data sources.

Derler: § 3.5 at p. 40 (attributes and links to documents stored in data sources are stored separately in a database, and as such constitute a connection cache).

Alternatively, a Slave Process receives a first request from a particular Web client, opens a connection to, and starts a session with, the Hyper-G server on behalf of the Web client. It maintains that connection to the Hyper-G server so that it can service all subsequent requests from the same client. Thus, the Slave Process caches the open connection to the Hyper-G server. The purpose of this is to avoid having to reestablish connections to the Hyper-G server each time, § 4.2.3.

Claim 5.

The computer-implemented method in claim 3 wherein said step of processing said request includes the step of logging into said one or more data sources.

Derler: § 3.3.3 at pp. 35-36 (access rights dictated by username and password, inherently teaches "logging in" to data source; § 4.5.6.1 at pp. 76-77; Figure 4.8 ("you have been logged into the IICM Information Server")).

Application/Control Number: 90/008,342 90/008,562
Art Unit: 3992 90/008,574

Page 39

Claim 6.

The computer-implemented method in claim 3 wherein said step of dynamically generating said Web page includes the step of maintaining a page cache containing said Web page.

Derler: § 3.3.5 at p. 37 (caching of documents (Web pages) in a cache; step of dynamically generating a Web page is performed by retrieving a cached document in response to a request for the document).

Claim 7.

The computer-implemented method in claim 3 wherein said page server includes custom HTML extension templates for configuring said Web page.

Derler: § 3.3.4 at p. 37 (Hyper-G server generates HTML elements (custom extension templates) configuring for use search results in form of a Web page; see generally § 4.5 and subsections; §§ 2.5, 2.5.4 (fill-out forms are "extensions of initial HTML specification"), §§ 4.6.1 (HTML files provide user interface), 4.6.1.1- 4.6.1.2; 4.6.

Claim 8.

The computer-implemented method in claim 7 wherein said step of processing said request further includes the step of inserting said dynamically retrieved data from said one or more data sources into said custom HTML extension templates.

Application/Control Number: 90/008,342 90/008,562
Art Unit: 3992 90/008,574

Page 40

Derler: Id., e.g., § 4.6.1.1; Figure 4.13 shows data retrieved from data source inserted into HTML extension template defining presentation of results of search.

Regarding Claim 9:

With regard to the limitations of claims under reexamination, an analysis must be made under 35 USC 112, sixth paragraph.

According to **MPEP 2181**:

A claim limitation will be presumed to invoke 35 U.S.C. 112, sixth paragraph, if it meets the following 3-prong analysis:

- (A) the claim limitations must use the phrase “means for” or “step for;”
- (B) the “means for” or “step for” must be modified by functional language; and
- (C) the phrase “means for” or “step for” must not be modified by sufficient structure, material, or acts for achieving the specified function.

Such a claim limitation is to be construed to cover the corresponding structure, described in the specification and equivalents thereof.

MPEP 2183 Making a Prima Facie Case of Equivalence

Factors that support a conclusion that the prior art element is an equivalent are:

Application/Control Number: 90/008,342 90/008,562
Art Unit: 3992 90/008,574

Page 41

(A) the prior art element performs the identical function specified in the claim in substantially the same way, and produces substantially the same results as the corresponding element disclosed in the specification.

(B) a person of ordinary skill in the art would have recognized the interchangeability of the element shown in the prior art for the corresponding element disclosed in the specification.

(C) there are insubstantial differences between the prior art element and the corresponding element disclosed in the specification.

(D) the prior art element is a structural equivalent of the corresponding element disclosed in the specification. That is, the prior art element performs the function specified in the claim in substantially the same manner as the function is performed by the corresponding element described in the specification.

In reference to USPN 5,894,554 to Lowery et al:

The cited function, "generating said request", is linked to a corresponding structure, "means for generating said requests."

The "means for generating a request" is described as a Web client machine running a Web browser. (Lowery: Col. 1:24-26; Col. 2:4-7; Col. 4:12-15; Col. 4:55-57; Col. 6:27-31; See Col. 8:25-29.) The Web Client Computer (the corresponding structure), as noted in the Specification, having a processor which operates a Web browser, is the corresponding structure that

Application/Control Number: 90/008,342 90/008,562

Page 42

Art Unit: 3992

90/008,574

accomplishes the function of generating a request. (a processor of a computer that is, or has, a Web client running a Web browser" or equivalents thereof).

The cited function, "receiving said request from said first computer", is linked to a corresponding structure, the "means for receiving said request from said first computer."

The "means for receiving said request from said first computer" is described by the Specification as the Web server 201 receiving requests from the Web client 200. (Lowery: Figure 4; Figure 5; Col. 4:54-59; See Figure 2; Col. 3:64-Col. 4:10.) The Web server 201 is also repeatedly described as including Web server executable, Id. As described above, the Specification also establishes that that even if a Web server is software, the software module operates on a computer as described within the specification. The corresponding structure of the "means for receiving" is "a processor of a computer that is, or has, a Web server running Web server executable" or equivalents thereof. The corresponding structure accomplishes the function of "receiving said request from said first computer."

The cited function, "processing said requests and dynamically generating a web page in response to said request" is linked to a corresponding structure, a "page server processing means processing said requests and dynamically generating a web page in response to said request."

With regard to the function of the "processing said requests and dynamically generating a web page in response to said request", the Specification shows that such function is accomplished by

Application/Control Number: 90/008,342 90/008,562
Art Unit: 3992 90/008,574

Page 43

a corresponding Page server 404(1)-(n) structure. (Lowery: Figure 4; Figure 5; Col. 5:37-Col. 6:31; Col. 8:39-43.) The Page server is page generation software. The page server software is operated on a computer system, a processor of a computer that runs Page server software/ page-generating software that generates a dynamic Web page, as described by the Specification. In one embodiment, Page servers reside on a separate machine to accomplish the claimed function. (Lowery: Col. 5 line 49-50.)

Claim 9:

[a] A networked system for managing a dynamic Web page generating request, said system comprising:

The Derler Thesis discloses a networked system for managing dynamic Web page generation requests from a WWW client sent to a Web server (WWW Gateway). Introduction, pp. 1-2; § 3.3.4 at p. 37; § 4.1 at pp. 43-44; § 4.7.2 at pp. 94-96; Figs. 4.16-4.17 at p. 95 (depicting networked system of WWW client, WWW Gateway, Hyper-G Server and remote servers (page servers) and document cache (page cache)).

[b] one or more data sources;

Derler: See id; Figure 4.17 (Hyper-G server, remote server and document cache are data sources.)

Application/Control Number: 90/008,342 90/008,562
Art Unit: 3992 90/008,574

Page 44

[c] a page server having a processing means;

Derler: "processing means" refers to a general-purpose processor or microprocessor 102 (Figure 1) and equivalents thereof, col. 3 lines 25 et seq. § 4.4.2, "The class WWWSlave contains all data and code necessary to process requests (slave process as a page server) from a single WWW client..." Alternately, The Hyper-G server is a page server (serves pages to Slave Process) and as such inherently has a general-purpose processor, the same as or equivalent to the processing means of the '554 patent. Derler, § 3.1 at p. 32; Figs. 4.16-4.17 and accompanying disclosure at pp. 94-95.

[d] a first computer system including means for generating said request; and

Derler: "means for generating said request" refers to a Web client in the form of a Web browser and equivalents thereof. Col. 4 lines 11-17. See Derler, § 1 at p. 1; § 3.3.4 at p. 37; Figs. 4.16-4.17 and accompanying disclosure at pp. 94-95 (depicting WWW Client, which is a first computer system which has a browser generating a Web page request). See also Derler at p. 4 ("WWW clients, often called 'WWW browsers'"); p. 48; pp. 67-68 (explaining user interaction with Hyper-G systems with "WWW clients like Mosaic or Netscape", both of which are Web browsers).

[el] a second computer system including

Application/Control Number: 90/008,342 90/008,562
Art Unit: 3992 90/008,574

Page 45

Derler: WWW Gateway is the second computing system. See generally Chapter 4.

[e2] means for receiving said request from said first computer,

Derler: The means for receiving request is Web server executable 201E (Figure 4) and equivalents thereof. Derler's Master process is a Web server executable process, same as or equivalent to '554 patent Web server executable. See Derler § 4.2.2 at p. 48-49; § 4.2.3 at p. 49-50; Figs. 4.1, 4.16-4.17 and accompanying disclosure at pp. 94-95 (depicting WWW Gateway and Master Process receives a Web page request from WWW client (first computer)).

[e3] said second computer system also including a router, said router routing said request from said second computer system to said page server,

Derler: See Id; see also Fig. 4.2, p. 52, route to passing process, p. 49-50, (alternately)

Child/Slave processes are a "router" within meaning of the '554 patent since they route request to Hyper-G page server.

[e4] wherein said routing further includes intercepting said request at said second computer,

Derler: § 4.2.2 at pp. 48-49; § 4.2.3 at pp. 49-50; Figs. 4.16-4.17 and accompanying disclosure at pp. 94-95; forking process of Master indicates interception of WWW request by "Passing Process".

Application/Control Number: 90/008,342 90/008,562
Art Unit: 3992 90/008,574

Page 46

[e5] routing said request from said second computer to a dispatcher, and

Request is routed from Master Server to Passing Process. Passing Process selects appropriate Slave Process and "dispatches" request to said Slave Process for processing. Derler: § 4.2.3 at pp. 49-50

[e6] dispatching said request to said page server

Derler: See id.; Figure 4.1, page server is Hyper-G server; Passing Process dispatches a Hyper-G request to the Slave Process (page server), which in turn may connect to a Hyper-G server. See also discussion of "dispatching" for element [b6] for claim 1, supra.

[e7] said page server receiving said request and

Derler: See 4.2.3, Slave Process receives request from Passing Process.

[e8] releasing said second computer system to process other requests,

Derler: § 4.2.2 at pp. 48-49; see also, e.g., Fig 4.5 and accompanying disclosure at pp. 49-54 (depiction of WWW Gateway processing multiple concurrent requests).

Application/Control Number: 90/008,342 90/008,562

Page 47

Art Unit: 3992

90/008,574

[e9] said page server processing means processing said request and dynamically generating a Web page in response to said request,

Derler: § 3.3.4 at p. 37; pp. 52-54 and Figs. 4.16-4.17 and accompanying disclosure at pp. 94-95 (the general-purpose processor ("processing means") in the Hyper-G page server and/or remote page servers process request). See also, e.g., § 4.4.2 at p. 65; Fig. 4.10 at p. 71 (sample of a dynamically-generated Web page in response to request). See also § 4.2.3 at p. 50: "When [a slave process (page server) on the WWW gateway] has accepted a connection initiated by the passing process, it reads the request, processes it (usually by transforming into a Hyper-G request for the server), receives the response from the Hyper-G server, transforms the response again if necessary, sends it to the passing process, and closes the connection to the passing process." Inherently, if the client receives a response from the Hyper-G server, the Hyper-G server has processed the request. (Content may be served from the Hyper-G server (page server), received and processed at the Slave process (page server), and in turn received at the Passing Process (page server), prior to being delivered as a request response to the client. Each of these processes may be involved in "serving pages".)

[e10] said Web page including data dynamically retrieved from said one or more data sources.

Derler: § 3.3.4 at p. 37; Fig. 4.10 at p. 71 (depicting sample dynamically-generated Web pages from multiple sources); data source can be data cache, see § 3.3.5; see p. 95 Figure 4.17, data source can be a remote server.

Application/Control Number: 90/008,342 90/008,562
Art Unit: 3992 90/008,574

Page 48

Claim 10:

[a] The networked system in claim 9 wherein said router in said second computer system includes:

Derler: § 4.2.3 at pp. 49-50 (Master Process, Slave Process, and Passing Process interact. The WWW Gateway "second computer system" involves the Master Process; Figs. 4.16-4.17 and accompanying disclosure at pp. 94-95.

[b] an interceptor intercepting said request at said second computer system and routing said request; and

Derler: § 4.2.2 at p. 48; § 4.2.3 at pp. 49-50 (master and forking process comprise an interceptor / Passing Process) As noted above, the Passing Process "intercepts" the request, and makes a determination as to which Slave Process to route the request.

[c] a dispatcher receiving said routed request from said interceptor and dispatching said request to said page server.

The Passing Process dispatches the request to the appropriate Slave Process (page server).

The Slave Process and the Hyper-G Server work together to serve pages. Derler: § 4.2.3 at pp. 49-50; Figs. 4.1-4.4

Application/Control Number: 90/008,342 90/008,562
Art Unit: 3992 90/008,574

Page 49

Claim 11:

[a] A machine readable medium having stored thereon data representing sequences of instructions, which when executed by a computer system, cause said computer system to perform the steps of:

Derler: The Hyper-G system is a computer system executing software instructions which are inherently stored in a machine readable medium. § 1 at p. 1; § 3.3.4 at p. 37; § 3.1 at p. 32. See also Fig. 4.16-4.17 at p. 95; § 4.6.3. See also generally Chapter 4 (WWW gateway is a computer system which includes software).

[bl] routing a dynamic Web page generation request from a Web server to a page server,

Derler: § 4.2.2 at p. 49; § 4.2.3 at pp. 50, 52, § 4.7.2 at p. 95; Figs. 4.16-17; Web page requests are dynamically routed from WWW Gateway ("Web server" / Master Process) to page server (Slave Process and Hyper- G server or a remote server work together to serve pages).

[b2] said page server receiving said request and

Derler: The client Web page request is received by the page server for processing. See § 4.2.3.

[b3] releasing said Web server to process other requests

Application/Control Number: 90/008,342 90/008,562
Art Unit: 3992 90/008,574

Page 50

Derler: Derler's WWW Gateway uses parallel processing of Web page requests wherein the WWW Gateway is released from one request to process other requests. See § 4.2.2 at pp. 48-49; § 4.2.3 at p. 49 (Master process of WWW Gateway), Figs. 4.1, 4.2, 4.3, 4.4 and 4.5 (pp. 52-55) illustrating process of releasing WWW Gateway to process other requests in parallel.

[b4] wherein said routing step further includes the steps of intercepting said request at said Web server,

Derler: § 4.2.2 at pp. 48-49; § 4.2.3 at p. 49; Figs. 4.16-4.17 and accompanying disclosure at pp. 94-95; forking process of Master indicates interception of WWW request by Passing Process.

[b5] routing said request from said Web server to a dispatcher, and

Derler: § 4.2.3 at pp. 49-50, Web Server routes request to Passing Process. Passing Process dispatches request to appropriate Slave Process. Fig. 4.7

[b6] dispatching said request to said page server;

Derler: See Id; Passing Process dispatches request to appropriate Slave Process (page server), which in turn may dispatch request to Hyper-G server (alternate page server).

Application/Control Number: 90/008,342 90/008,562
Art Unit: 3992 90/008,574

Page 51

[c] processing said request, said processing being performed by said page server while said Web server concurrently processes said other requests; and

Derler: § 4.2.2 at pp. 48-49; § 4.2.3 at p. 49, pp. 52-54, processing of request by Slave Process (page server); Fig. 4.7, such processing occurs concurrently with the WWW Gateway processing other requests. See also § 4.2.3 at p. 50 ("When it [a Slave process on the WWW Gateway] has accepted a connection initiated by the passing process, it reads the request, processes it (usually by transforming it into a Hyper-G request for the server, receives the response from the Hyper-G server, transforms the response again if necessary, sends it to the passing service, closes the connection to the passing process.") The client receives a response from Passing Process (Fig.4.7), after the Slave Process (page server) and possibly with further processing by the Hyper-G server (alternate page server), has processed the request.

"After accepting the connection, the Master forks generating a child process. Immediately after that, the master closes its connection to W3client A and is then ready to accept other connections (Web server concurrently processes said other requests)."

[d] dynamically generating a Web page, said Web page including data retrieved from one or more data sources.

Derler: § 3.3.4 at p. 37; Fig. 4.10 at p. 71 (depicting sample dynamically-generated Web page from data from one or more sources); data cache and remote server are data sources, § 3.3.5 and p. 95.

Application/Control Number: 90/008,342 90/008,562
Art Unit: 3992 90/008,574

Page 52

Claims 1-4 and 7-11 are rejected under 35 U.S.C. 102(a) as being anticipated by Dienst:

Claim 1:

[a] A computer-implemented method for managing a dynamic Web page generation request to a Web server, said computer-implemented method comprising the steps of:

Lagoze describes a Web-based architecture, called "Dienst," that provides access to distributed document libraries over the World WideWeb (Web / WWW). (See, Abstract at p. 1.) The architecture comprises a Web server, a CGI stub, a Dienst server and a document database (§ 4.1 at p. 9). Using a Web browser, a user can submit a search request for certain documents meeting the search parameters. A so-called "Dienst request" is packaged within the path portion of a standard HTTP request and transmitted from the user's Web client (browser) to the Web server. (§4.1 at p. 11). "Search results are presented as a hypertext document," i.e., a Web page, which is transmitted back to the client as a standard HTTP response. (§ 1 at p. 6; § 4.1 at p. 11.) Because of the dynamic nature of the results, each search request constitutes a "dynamic Web page generation request." The Dienst Manual describes a computer-implemented method for handling dynamic Web page requests. (P. 6; pp. 9-11, Fig. 2.) Since the Web clients accessing the Dienst system use "popular browsers of the World Wide Web [such] as Mosaic, Cello and Netscape" (p. 6), the information presented to the Web clients constitutes "Web pages."

[bl] routing said request from said Web server to a page server,

Application/Control Number: 90/008,342 90/008,562
Art Unit: 3992 90/008,574

Page 53

Dienst Manual: (Pp. 10-11; Fig. 2); Web server is WWW server; page server is Dienst Server.

Arrows in Figure 2 illustrate routing of request from WWW server to Dienst Server.

(§ 4.1 at p. 10) As described in the Lagoze reference, when the Web server detects a Dienst request in the path portion of an HTTP request received from a Web client, the Web server invokes (routes request through CGI Stub) the CGI stub. The CGI stub "is a small, simple executable that strips the Dienst request from the HTTP request, packages a few important environment variables that contain information about the request and sends (further routing said request) them via socket I/O to the Dienst server." (§ 4.1 at pp. 11.) The Dienst server is a "page server" in that it processes the Dienst request; formulates the results of a search, and transmits a dynamic Web page containing the search results back to the Web client. (Id.)

[b2] said page server receiving said request and

Dienst Manual: The Dienst Server receives the request. (See Id.)

[b3] releasing said Web server to process other requests,

Dienst Manual: (P. 11.) CGI stub strips Dienst request from HTTP request; the CGI stub is invoked each time the gateway Web server receives a Dienst request. Inherently, the WWW server is released from processing that request, and can process other requests. This is further apparent from nature of the Dienst system to provide for simultaneous access to documents on

Application/Control Number: 90/008,342 90/008,562
Art Unit: 3992 90/008,574

Page 54

Dienst system for distributed WWW clients. P. 11, The Web server serves as a front-end for Dienst. The Web server is set up as a gateway to a Dienst server.

[b4] wherein said routing step further includes the steps of intercepting said request at said Web server,

Dienst Manual: As described above [b1], the Web server recognizes from the path portion of the HTTP request that the request is a Dienst request (i.e., a dynamic Web page generation request) and passes the request to the CGI stub, thereby "intercepting" it. That is, the Web server stops or interrupts its processing of the Dienst request and deflects processing of the request to the CGI stub. (P. 11)

[b5] routing said request from said Web server to a dispatcher, and

Dienst Manual: Request is passed to CGI stub (routing process) (Fig. 2; pp. 10-11), dispatching is a function of Dienst CGI stub. The CGI stub serves as a dispatcher that dispatches the request to one or more Dienst servers. (§ 4.1 and 4.2 at pp. 11-12.)

[b6] dispatching said request to said page server;

Dienst Manual: (P. 11); Dienst CGI stub sends (dispatches) request via I/O socket to Dienst Server (page server). (P. 12), The Dienst Manual describes a Dienst Server which functions as a

Application/Control Number: 90/008,342 90/008,562

Page 55

Art Unit: 3992 90/008,574

user interface server which "provides a front-end for searching the collection provided by all inter-operating Dienst servers." See Figure 2, page 10 "Dienst Server" and "Other Dienst Servers". The "other Dienst servers" host a distributed collection of documents. (P. 12), "Each server is able to locate the indexing and repository site for a specific publisher using tables that are maintained by a distinguished server... The respective user interface server then dispatches search requests, in parallel to the Dienst index servers that corresponds to the publisher selections(s)..."(Dienst Manual, page 12, ¶¶ 1, 2.)

The CGI Stub dispatches to the Dienst Server, which may in turn "dispatch" other Dienst Servers or to document databases. (See Fig. 2, p. 10)

[c] processing said request, said processing being performed by said page server while said Web server concurrently processes said other requests; and

Dienst Manual: Fig. 2; pp. 10-11; function of Dienst Server is to process requests and generate Web pages. Because a "Dienst server is a standalone multi-threaded process" (§ 4.1 at p. 11), it necessarily processes the request while the Web server continues to concurrently process other HTTP requests. P. 11, The Web server serves as a front end for Dienst. Inherently, with the CGI stub program intercepting the Dienst request, the Web server process is released from processing that request, and can process other requests concurrently. This is further apparent from the nature of the Dienst system to provide for simultaneous access to documents on the Dienst system from distributed WWW clients.

Application/Control Number: 90/008,342 90/008,562
Art Unit: 3992 90/008,574

Page 56

[dl] dynamically generating a Web page in response to said request,

Dienst Manual: "Search results are presented as a hypertext document," i.e., a Web page, which is transmitted back to the client as a standard HTTP response. (§ 1 at p. 6); "responses from Dienst are formatted" (§ 4.1 at p. 11); The function of Dienst server is to dynamically generate a Web page in response to request; (see also Figs. 4 and 5, pp. 14-15) (showing dynamically generated pages); (p. 58) (describing information that can be provided on dynamically-generated pages).

[d2] said Web page including data dynamically retrieved from one or more data sources.

Dienst Manual: Pp. 6, 9-12, Fig. 2, data sources are repositories of documents on Dienst Server or in a document database; data is dynamically retrieved at time of processing of request. See also Figs. 4 and 5, pp 9-11 & 58.

Claim 2:

The computer-implemented method in claim 1 herein said step of processing said request includes the step of identifying^o said one or more data sources from which to retrieve said data.

Dienst Manual: "A user of a Dienst server perceives a single logical collection [of documents], even though the collection is distributed over multiple servers. This is accomplished by

Application/Control Number: 90/008,342 90/008,562

Page 57

Art Unit: 3992

90/008,574

interaction between a set of Dienst servers " (§ 4.2 at p. 11.) "Each server, in a set of interoperating Dienst servers, is capable of responding to a request for any document in the distributed collection. The respective server uses the publisher in the docid of the requested document and the mapping tables (processing / identifying one or more data sources) downloaded from the meta server to route the document request to the server that acts as the repository for the publisher. This routing is transparent to the user." (Id. at p. 12.) Pp. 11-12, indexing service and distributed searching function of Dienst Server inherently identifies a data source from which to retrieve data since that is the function of the indexing and searching functions.

Claim 3:

The computer-implemented method in claim 2 wherein said step of dynamically generating said Web page includes the step of dynamically retrieving said data from said one or more data sources.

Dienst Manual: P. 11; Dienst server dynamically retrieves data from data source such as a repository of documents. (Figure 2, Document Database) P. 12, distributed document access feature of Dienst server. See also Figs. 4 and 5 (showing pages with dynamically retrieved data).

Claim 4:

Application/Control Number: 90/008,342 90/008,562

Page 58

Art Unit: 3992

90/008,574

The computer-implemented method in claim 3 wherein said step of processing said request includes the step of said page server maintaining a connection cache to said one or more data sources.

Dienst Manual: As mentioned above, in the Dienst architecture, "[a] Dienst server perceives a single logical collection [of documents], even though the collection is distributed over multiple servers. This is accomplished by interaction between a set of Dienst servers" (§ 4.2 at p. 11.) Each document in the document database has an associated "docid" and "each docid ... consists of a publisher identifier and a name." (§ 4.2 at p. 12.), "Each server in a set of interoperating Dienst servers indexes and acts as a repository for documents from one or more publishers, in the current version, all the documents for a publisher must be indexed and reside on the same server. Each server is able to locate the indexing and repository site for a specific publisher using tables that are maintained by a distinguished server, the meta server This allows every server to provide the next two functions distributed document access - Each server, in a set of interoperating Dienst servers, is capable of responding to a request for any document in the distributed collection. The respective server uses the publisher in the docid of the requested document and the mapping tables downloaded from the meta server to route the document request to the server that acts as the repository for the publisher. This routing is transparent to the user." (Id. at p. 12.) Thus, the meta server serves as a "connection cache" in that it maintains information about which Dienst server in a set of interoperating Dienst servers is the repository (i.e., data source) for a given publisher's documents. (Pp. 9, 11-12); server registration process, and/or indexing service, is maintenance of connection cache to data source.

Application/Control Number: 90/008,342 90/008,562
Art Unit: 3992 90/008,574

Page 59

Claim 7:

The computer-implemented method in claim 3 wherein said page server includes custom HTML extension templates for configuring said Web page.

Dienst Manual: P. 6, "Search results are presented as a hypertext document. (configured Web page)" (Pp. 15, 17, 60 and 61) HTML is used for configuring Web pages, e.g., pages containing data in response to queries of Dienst system; HTML image map templates of Fig. 7 (p. 17) are custom HTML extension templates. (See also Figures 4 and 5, pp. 14, 15.)

The Dienst server provides a Web designer who is managing the document database with a variety of document format templates that can be used to provide different formats for display of documents retrieved from the database. These templates include "inline - a paged format that is an image displayed inline on HTML pages" as well as "composite - a paged format that is a composite of thumbnail images." (§ 7 at pp. 33-35; § 5 at p. 17.) With the composite format, "[each] thumbnail page is implemented using a HTML image map, allowing the user to select on a page and dispatch a URL to view that specific page in readable form." § 5 at p. 17.

Claim 8:

The computer-implemented method in claim 7 wherein said step of processing said request further includes the step of inserting said dynamically retrieved data from said one or more data sources into said custom HTML extension templates.

Application/Control Number: 90/008,342 90/008,562
Art Unit: 3992 90/008,574

Page 60

Dienst Manual: P. 17, Fig. 7, disclosing HTML image map templates which are custom HTML extension templates; data retrieved from data sources are inserted into HTML image map templates. See also Figs. 4 and 5, which appear to be based on HTML templates.

When document data is retrieved from the document database it is inserted into one of the document format templates supported by the Dienst server for display to the user. (§ 7 at pp. 33-35; § 5 at p. 17.)

Claim 9:

[a] A networked system for managing a dynamic Web page generation request, said system comprising:

Dienst Manual: (Abstract; pp. 5-6, 9-11); Fig. 2 -showing distributed networked system for managing dynamic Web page generation requests). A Web-based architecture, called "Dienst," that provides access to distributed document libraries over the World Wide Web (Web). (See, Abstract at p. 1.) The architecture comprises a Web server, a CGI stub, a Dienst server and a document database. (§ 4.1 at p. 9.) Using a Web browser, a user can submit a search request for certain documents meeting the search parameters. A so-called "Dienst request" is packaged within the path portion of a standard HTTP request and transmitted from the user's Web client to the Web server. (Lagoze, §4.1 at p. 11.) "Search results are presented as a hypertext document," i.e., a Web page, which is transmitted back to the client as a standard HTTP response. (§ 1 at p. 6; § 4.1 at p. 11.) The Dienst Manual describes a computer-implemented method for handling

Application/Control Number: 90/008,342 90/008,562
Art Unit: 3992 90/008,574

Page 61

dynamic Web page requests. The Dienst system includes a WWW server which manages dynamic Web page requests from WWW clients, and a page server ("Dienst Server"); system is a computer-implemented method as indicated throughout the document. Because of the dynamic nature of the results, each search request constitutes a "dynamic Web page generation request." The Web clients accessing Dienst system use "popular browsers of the World Wide Web [such] as Mosaic, Cello and Netscape" (p. 6), and the information presented to the Web clients constitutes "Web pages".

See comments above regarding 35 U.S.C. § 112, ¶ 6.

[b] one or more data sources;

Dienst Manual (Pp. 6 and 9-12); Dienst servers storing documents in the Dienst system are data sources, as are document databases. The Dienst architecture includes a "document database" consisting of one or more document repositories (data sources). (§ 4.1 at p. 9.) See Fig. 2, p. 10.

[c] a page server having a processing means;

Dienst Manual: Under § 112, ¶ 6, "processing means" refers to a general-purpose processor or microprocessor 102 ('554: Fig. 1) and equivalents thereof ('554": col. 3 lines 25 et seq.) See Dienst Manual (pp. 6 and 9- 12) and Fig. 2. ; Dienst Servers are computers which inherently

Application/Control Number: 90/008,342 90/008,562
Art Unit: 3992 90/008,574

Page 62

include general-purpose processors or equivalent thereof. The Dienst server is a "page server" and it necessarily includes a processing means.

[d] a first computer system including means for generating said request; and

Dienst Manual: Under § 112, ¶ 6, "means for generating said request" refers to a Web client in the form of a Web browser. ('554: Col. 4 lines 11-17.) See Dienst Manual, WWW client, Fig. 2, p. 6 (mentioning popular Web browsers such as Netscape) and p. 10. Using a Web browser, a user can submit a search request for certain documents meeting the search parameters. The user's Web browser constitutes a "first computer system."

[e1] a second computer system including

Dienst Manual: WWW server; Fig. 2; pages 10 and 11. The Web server of the Dienst architecture constitutes a "second computer system" and it is intended to receive requests from client Web browsers. "A Web server that serves as a front end for Dienst..." (p. 11)

[e2] means for receiving said request from said first computer,

Dienst Manual: Under § 112 ¶ 6, the means for receiving request is Web server executable 201E ('554: Figure 4) and equivalents thereof. Dienst WWW server includes an executable as a means for receiving request from WWW client. Fig. 2, pages 10 and 11 (CERN & NCSA servers), 31

Application/Control Number: 90/008,342 90/008,562
Art Unit: 3992 90/008,574

Page 63

and 32 (describing downloading/ installing NCSA or CERN Web server code (executable)).

Executable server code on a processor provides the means for receiving a request.

[e3] said second computer system also including a router, said router routing said request from said second computer system to said page server,

Dienst Manual: (Fig. 2; pp. 10-11); Dienst CGI stub acts as a router routing request from WWW server to Dienst Server. As described above, when the Web server detects a Dienst request in the path portion of an HTTP request received from a Web client, the Web server (second computer) invokes the CGI stub. The CGI stub "is a small, simple executable that strips the Dienst request from the HTTP request, packages a few important environment variables that contain information about the request and sends (routes) them via socket I/O to the Dienst server (page server)." (§ 4.1 at pp. 11.) The Dienst server is a "page server" in that it processes the Dienst request, formulates the results of a search, and transmits a dynamic Web page containing the search results back to the Web client (Id.).

[e4] wherein said routing further includes intercepting said request at said second computer,

Dienst Manual: As described above, the Web server (second computer) recognizes (intercepts) from the path portion of the HTTP request that the request is a Dienst request (i.e., a dynamic Web page generation request) and passes the request to the CGI stub (intercepting at second computer / Web server).

Application/Control Number: 90/008,342 90/008,562
Art Unit: 3992 90/008,574

Page 64

[e5] routing said request from said second computer to a dispatcher, and

Dienst Manual: Pp. 10-11, routing request from Web server to CGI stub / dispatcher. P. 11; Web server (second computer) recognizes (intercepts) from the path portion of the HTTP request that the request is a Dienst request (i.e., a dynamic Web page generation request) and passes (route from second computer to dispatcher) the request to the CGI stub (intercepting at second computer / Web server) ("the stub is a small, simple executable that strips the Dienst request from the HTTP request."). The CGI stub serves as a dispatcher that dispatches the request to one or more Dienst servers. (§ 4.1 and 4.2 at pp. 11-12.)

[e6] dispatching said request to said page server,

Dienst Manual P. 11; CGI stub send (dispatches) request via I/O socket to Dienst Server (page server). The CGI stub serves as a dispatcher that dispatches the request to one or more Dienst servers. (§ 4.1 and 4.2 at pp. 11-12.) See additional "dispatching" comments at claim 1 [b6], supra.

[e7] said page server receiving said request and

Dienst Manual: (Fig. 2; pp. 10 and 11); Dienst Server receives request.

Application/Control Number: 90/008,342 90/008,562
Art Unit: 3992 90/008,574

Page 65

[e8] releasing said second computer system to process other requests,

Dienst Manual: P. 11. CGI stub strips Dienst request from HTTP request; the CGI stub is invoked each time the gateway Web server (second computer) receives a Dienst request.

Inherently, the WWW server (second computer) is released from processing that request, and can process other requests. "A Web server that serves as a front-end for Dienst...", (p. 11).

[e9] said page server processing means processing said request and dynamically generating a Web page in response to said request,

Dienst Manual: Fig. 2; pp. 9-12; function of general-purpose processor ("processing means"), in the Dienst Server is to process requests and generate Web pages. See also Figs. 4 and 5, pp. 14-15 (showing dynamically generated pages); p. 58 (describing information that can be provided on dynamically-generated pages). A Dienst server retrieves information from the document database that meets a user's search criteria and generates a Web page containing the results of the search. (§ 1 at p. 6; § 4.1 at p. 11) ("Search results are presented as a hypertext document," i.e., a Web page, which is transmitted back to the client as a standard HTTP response). Because each search can be different, the process of responding to search requests necessarily is "dynamic."

[e10] said Web page including data dynamically retrieved from said one or more data sources.

Application/Control Number: 90/008,342 90/008,562

Page 66

Art Unit: 3992

90/008,574

Dienst Manual: (Pp. 6, 9-12), Fig. 2, data sources are repositories of documents on Dienst Server or in a document database; data is dynamically retrieved at time of processing of request. (See also Figs. 4 and 5, pp. 14-15) -showing dynamically generated pages); (p. 58)-describing information that can be provided on dynamically generated pages.

Claim 10:

[a] The networked system in claim 9 wherein said router in said second computer system includes:

Dienst Manual: As described above, the Web server (second computer) recognizes from the path portion of the HTTP request that the request is a Dienst request (i.e., a dynamic Web page generation request) and passes (routes) the request to the CGI stub. (Pp. 10-11; Fig. 2); WWW server (second computer) functions as a router to CGI Stub.

[b] an interceptor intercepting said request at said second computer system and routing said request; and

Dienst Manual: (Id.) – Fig. 2, WWW Server (second computer) intercepts / routes information to CGI stub for further routing to Dienst server (page server). Interception is triggered by a “Dienst request” (p. 11) packaged within a path portion of the HTTP request. When the HTTP request contains a “Dienst request” the CGI stub program is invoked.

Application/Control Number: 90/008,342 90/008,562

Page 67

Art Unit: 3992 90/008,574

[c] a dispatcher receiving said routed request from said interceptor and dispatching said request to said page server.

Dienst: (Id.), dispatching request to a page server is a feature of CGI-stub and thus it functions as a "dispatcher." See also discussion of "dispatching" claim 1[b6], supra. See Fig. 2, WWW Server (interceptor) routes request to CGI Stub. CGI Stub (dispatcher) dispatches request to Dienst Server (page server). (§ 4.1 and 4.2 at pp. 11-12.)

Claim 11:

[a] A machine readable medium having stored thereon data representing sequences of instructions, which when executed by a computer system, cause said computer system to perform the steps of:

Dienst: Pp. 6, 9-11, 30-33 (computer system comprises WWW server which includes a software Web server process such as CERN or NCSA Web server plus Dienst Server which is a multi-threaded software processes, such processes are inherently sequences of instructions stored on a machine- readable medium). See Fig. 2.

[bl] routing a dynamic Web page generation request from a Web server to a page server,

Dienst: Pp. 10-11; Fig. 2; Web page requests are routed from WWW server (Web server) to Dienst Server (page server), via a CGI Stub. The "Dienst request" packaged within the pat

Application/Control Number: 90/008,342 90/008,562

Page 68

Art Unit: 3992

90/008,574

portion of the HTTP request to the Web server (p. 11) triggers the routing of the dynamic Web page generation request (Dienst request) from a Web server to a page server (Dienst server).

[b2] said page server receiving said request and

Dienst: Fig. 2, pages 10 and 11, Dienst Server (page server) receives request.

[b3] releasing said Web server to process other requests wherein said routing step further includes the steps of

Dienst: P. 11, CGI stub strips Dienst request from HTTP request; the CGI stub is invoked each time the gateway Web server receives a Dienst request. Inherently, the WWW server is released from processing that request, and can process other requests. This is further apparent from the nature of the Dienst system to provide for simultaneous access to documents on the Dienst system for distributed WWW clients. The CGI stub accepts the request and forwards it to the Dienst server. Inherently, the Web server process is released from processing that request and can process other requests. "A Dienst server is a standalone multi-threaded process written in Perl." (§ 4.1 at p. 11.) A Web server that serves as a front-end for Dienst...", p. 11.

[b4] intercepting said request at said Web server,

Application/Control Number: 90/008,342 90/008,562
Art Unit: 3992 90/008,574

Page 69

Dienst: See Fig. 2, The Web server recognizes/ detects a path portion of an HTTP request, intercepts the request, and diverts it to a CGI stub.

[b5] routing said request from said Web server to a dispatcher, and

Dienst: Fig. 2, pp. 10-11, The HTTP request, detected by the Web Server is routed to the CGI Stub. The Dienst CGI stub (dispatcher) further sends request via I/O socket to Dienst Server.

[b6] dispatching said request to said page server;

Dienst: P. 11; CGI-stub dispatches request to a Dienst Server (page server). (§ 4.1 and 4.2 at pp. 11- 12.)

[c] processing said request, said processing being performed by said page server while said Web server concurrently processes said other requests; and

Dienst: Fig. 2, pp. 10-11; the function of the Dienst Server is to process requests and generate Web pages. Inherently, with the CGI stub program intercepting the Dienst request, the Web server process is released from processing that request, and can process other requests concurrently. The Dienst server processes the Dienst request. Because a "Dienst server is a standalone multi-threaded process" (§ 4.1 at p. 11), it necessarily processes the request while the Web server continues to concurrently process other HTTP requests. This is further apparent from

Application/Control Number: 90/008,342 90/008,562

Page 70

Art Unit: 3992

90/008,574

nature of the Dienst system to provide for simultaneous access to documents on Dienst system for distributed WWW clients. "A Web server that serves as a front-end for Dienst...", p. 11.

[d] dynamically generating a Web page, said Web page including data retrieved from one or more data sources.

Dienst: (Abstract; pp. 6, 9-12); Fig. 2; Dienst server dynamically generates Web page, retrieves data from data sources, such as documents on Dienst server or in a document database. See also Figs. 4 and 5, pp. 14-15 (showing dynamically generated pages); p. 58 (describing information that can be provided on dynamically-generated pages). See pages 57-61 regarding types of responses generated (generated Web pages) following a request. Because each search can be different, the process of responding to search requests necessarily is "dynamic."

Claim Rejections - 35 USC § 103

The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negated by the manner in which the invention was made.

The factual inquiries set forth in *Graham v. John Deere Co.*, 383 U.S. 1, 148 USPQ 459 (1966), that are applied for establishing a background for determining obviousness under 35 U.S.C. 103(a) are summarized as follows:

1. Determining the scope and contents of the prior art.

Application/Control Number: 90/008,342 90/008,562
Art Unit: 3992 90/008,574

Page 71

2. Ascertaining the differences between the prior art and the claims at issue.
3. Resolving the level of ordinary skill in the pertinent art.
4. Considering objective evidence present in the application indicating obviousness or nonobviousness.

Claims 5 and 6 are rejected under 35 U.S.C. 103(a) as being unpatentable over Dienst, in view of Derler.

Claim 5:

The computer-implemented method in claim 3 wherein said step of processing said request includes the step of logging into said one or more data sources.

Dienst: Fig. 2, The Dienst Server “logs into” one or more data sources (other Dienst Servers or Document Database) § 8.3.1 at p. 41, “The basic Dienst server interoperates with a distributed network of other Dienst servers...A user may submit searches to a number of publishers, which are indexed by the distributed sites. A search is processed in parallel by each of these sites (logged into one or more data sources) and uniform search results (processed request) are returned to the user. The foreign server package enhances searching by allowing a Dienst server to interoperate with non-Dienst technical report servers (logged into one or more data sources). (inherent to accessing documents from foreign server is logging into the foreign server).”

Even more explicitly, Derler discloses “logging into data sources.”

Derler: § 3.3.3 at pp. 35-36 (access rights dictated by username and password, inherently teaches “logging into” one or more data sources); § 4.5.6.1 at pp. 76-77, “...user has an account...logged

Application/Control Number: 90/008,342 90/008,562
Art Unit: 3992 90/008,574

Page 72

in as an identified Hyper-G user, which can be verified...Identification extends the user's access rights..."

Modification of the Dienst system to include securely logging into a data source as part of processing of requests, as taught by Derler, would be an obvious feature to access distributed documents.

Claim 6:

The computer-implemented method in claim 3 wherein said step of dynamically generating said Web page includes the step of maintaining a page cache containing said Web page.

Dienst discloses (p. 6 & 17) thumbnail images (page cache containing said Web page). The thumbnails are implemented using the HTML imagemap mechanism.

More explicitly Derler discloses caching of Web pages in a Web server to improve performance. (§ 3.3.5 at p. 37.)

Maintaining a page cache containing a Web page would be an obvious modification of the Dienst system. A person skilled in the art would have been motivated to add page caching in the Dienst server so as to provide ready access to frequently-requested documents without having to query foreign servers, as explained in the Derler Thesis at 3.3.5.

Application/Control Number: 90/008,342 90/008,562
Art Unit: 3992 90/008,574

Page 73

The following rejections are incorporated by reference as follows:

Claim Rejections - 35 USC § 102

(e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

Claims 1-11 are rejected under 35 U.S.C. 102(e) as being anticipated by USPN 5,701,451 to Rogers et al. See 90 / 008342, RXOSUB.R (11/27/2006) p. 3-7.

Claims 1-3, 5, and 7-11 are rejected under 35 U.S.C. 102(e) as being anticipated by USPN 6,249,291 B1 to Popp et al. See 90 / 008574, RXOSUB.R (04/03/2007) p. 73-78.

Claim Rejections - 35 USC § 102

(a) the invention was known or used by others in this country, or patented or described in a printed publication in this or a foreign country, before the invention thereof by the applicant for a patent.

Claims 1-5 and 7-11 are rejected under 35 U.S.C. 102(a) as being anticipated by Illustra.

The rejections are incorporated by reference to 90/008574, RXOSUB.R (04/03/2007), p. 50-55)

Claims 1-5 and 7-11 are rejected under 35 U.S.C. 102(a) as being anticipated by

Clausnitzer. The rejections are incorporated by reference to 90/008574, RXOSUB.R (04/03/2007), p. 61-66)

Application/Control Number: 90/008,342 90/008,562
Art Unit: 3992 90/008,574

Page 74

VI. Conclusion

In order to ensure full consideration of any amendments, affidavits or declarations, or other documents as evidence of patentability, such documents must be submitted in response to this Office action. Submissions after the next Office action, which is intended to be a final action, will be governed by the requirements of 37 CFR 1.116, after final rejection and 37 CFR 41.33 after appeal, which will be strictly enforced.

Any paper filed with the USPTO, i.e., any submission made, by either the Patent Owner or the Third Party Requester must be served on every other party in the reexamination proceeding, including any other third party requester that is part of the proceeding due to merger of the reexamination proceedings. As proof of service, the party submitting the paper to the Office must attach a Certificate of Service to the paper, which sets forth the name and address of the party served and the method of service. Papers filed without the required Certificate of Service may be denied consideration. See 37 CFR 1.550(f)

Please mail any communications to:

Attn: Mail Stop "Ex Parte Reexam"
Central Reexamination Unit
Commissioner for Patents
P. O. Box 1450
Alexandria VA 22313-1450

Please FAX any communications to:

(571) 273-9900
Central Reexamination Unit

Please hand-deliver any communications to:

Customer Service Window

Application/Control Number: 90/008,342 90/008,562
Art Unit: 3992 90/008,574

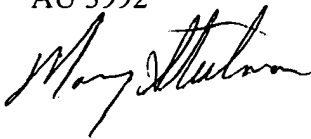
Page 75

Attn: Central Reexamination Unit
Randolph Building, Lobby Level
401 Dulany Street
Alexandria, VA 22314

Any inquiry concerning this communication or earlier communications from the examiner, or as to the status of this proceeding, should be directed to the Central Reexamination Unit at telephone number (571) 272-7705.

Mary Steelman
Primary Examiner

AU 3992



Conferees:


ESK

Exhibit 28

**IN THE UNITED STATES DISTRICT COURT
FOR THE DISTRICT OF DELAWARE**

**ORACLE CORPORATION and
ORACLE U.S.A. INC.,**

Plaintiffs,

v.

EPICREALM LICENSING, LLC,

Defendant.

C.A. No. 06-cv-414 (SLR)

AND RELATED COUNTERCLAIMS

**REPORT OF PLAINTIFFS' EXPERT MICHAEL I. SHAMOS, PH.D, J.D.
CONCERNING INVALIDITY**

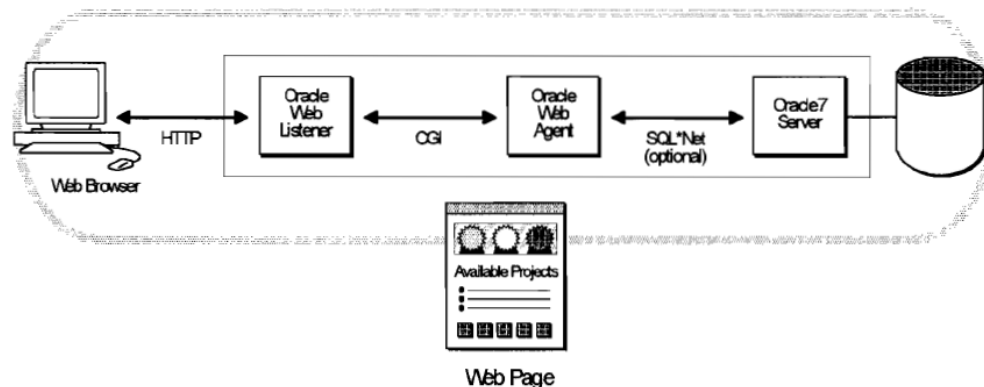
BACKGROUND & QUALIFICATIONS

1. My name is Michael I. Shamos. I hold the title of Distinguished Career Professor in the School of Computer Science at Carnegie Mellon University in Pittsburgh, Pennsylvania. I was a founder and Co-Director of the Institute for eCommerce at Carnegie Mellon and I now direct a graduate degree program in eBusiness Technologies. My résumé is attached as Exhibit 2 to this report.

2. I currently teach graduate courses at Carnegie Mellon in Electronic Commerce, including eCommerce Technology, Electronic Payment Systems, Electronic Voting and eCommerce Law and Regulation and have done so since 1999. In Fall 2007 I taught a new course entitled Law of Computer Technology.

221. Reference 153 is 248 pages long. The entire Bookman patent, including drawings, is only 9 pages long. There are numerous disclosures in reference 153 that are not present in Bookman. Bookman, for example, makes no mention of SQL*Net, which, as seen below, is expressly disclosed in reference 153 as being an optional component. SQL*Net utilizes load-balancing dispatching, which is not taught in Bookman. Reference 153 teaches TCP releasing, according to epicRealm's contention, which is not taught in Bookman. While concurrency is inherent in Bookman (simultaneous processing by the web server and page server) it is expressly taught in Reference 153. Reference 153 expressly teaches multiple data sources, while Bookman teaches one database. SQL*Net, available with Oracle 1.0 as described in Reference 153, used connection caching, which is not taught in Bookman. Reference 153 teaches page caching; Bookman does not. Bookman teaches updating a data source as part of a request inherently, through SQL. Reference 153 teaches updating expressly (MAC003112). Thus the teachings of Bookman and Reference 153 are considerably different.

222. The Oracle WebServer structure is also discussed in more detail below in connection with '335 claim 15. A diagram of its overall architecture, from page MAC003053, is reproduced below.



223. Briefly, the web server in Oracle is called the web listener, which receives requests. Requests for dynamic pages are intercepted and dispatched by a web agent to Oracle7 Server (page server), which accesses the necessary SQL databases to retrieve dynamic information used to configure the web page.

547. According to Mr. Lönnroth, upon its release on or before March 1, 1995, the WOW Gateway was put into use at various locations inside and outside the United States. Lönnroth Aff. at ¶¶ 7-8. After completing the code for the WOW Gateway, Joseph Pistritto and Mr. Lönnroth became involved in developing the Oracle WebServer, which was an enterprise-class package of software that was the commercial successor to the freely available WOW Gateway. Lönnroth Aff. at ¶ 9, Montinola Depo at 15:2-10, and ORCL01043070-091. Other members of the Oracle team that developed the Oracle WebServer included Katrina Montinola, Matthew Bookman, Brian Zuzga, Mala Anand, Kelly Ireland, and Aileen Jaitin. Montinola Depo at 15:11-21. The Oracle WebServer includes a Web Listener and a Web Agent and was designed to operate with the Oracle7 database server to provide access to database information in HTML format over the World Wide Web. This is described in the Oracle 1.0 References.

548. The Oracle WebServer together with the Oracle7 Database Server comprised a system for providing access to database information over the Web. See, generally, Oracle WebServer User's Guide (Release 1.0) at ORCL000002-224. Like the Oracle WOW Gateway, the Web Agent component of the Oracle WebServer provides the connectivity between a web server and a database server. Particularly, the Web Agent connects the Web Listener, a web server, to page generating applications residing on an Oracle7 database server. For example, a request is received at the Web Listener, which is a web server capable of handling requests for static or dynamic web pages. Requests for dynamic web pages are intercepted at the Web Listener by the Web Agent. The Web Agent examines such a request to determine which application on the database server should process the request. The Web Agent then logs into the Oracle7 database server and dispatches the request to the appropriate application. The appropriate application (i.e., the page server) executes to query the database and dynamically generate an HTML web page including the information sought by the request. This process is designed to operate asynchronously, so that while the application on the database server is processing a request, the Web Listener is free to process and respond to other requests. Lönnroth Aff. at ¶ 11.

Exhibit 29

Oracle[®] WebServer User's Guide

Release 1.0
Part No. A34986-2

ORACLE[®]

MAC003037

ORCL00941713

Oracle WebServer User's Guide, Release 1.0

Part No. A34986-2

Copyright © 1995 Oracle Corporation

All rights reserved. Printed in the U.S.A.

Contributing Authors: Mala Anand, Matthew Bookman, Martin Gruber,
Magnus Lonnroth, Katrina Montinola, Mary Owen, and Joe Pistrutto

Contributors: Arthur Gleckler, Walter Hudson, Kelly Ireland, Aileen Jaitin, Max Pollak, Sandy Venning, and Marcel Weekes

This software was not developed for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It is the customer's responsibility to take all appropriate measures to ensure the safe use of such applications if the programs are used for such purposes.

This software/ documentation contains proprietary information of Oracle Corporation; it is provided under a license agreement containing restrictions on use and disclosure and is also protected by copyright law. Reverse engineering of the software is prohibited.

If this software/ documentation is delivered to a U.S. Government Agency of the Department of Defense, then it is delivered with Restricted Rights and the following legend is applicable:

Restricted Rights Legend Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of DFARS 252.227-7013, Rights in Technical Data and Computer Software (October 1988).

Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065.

If this software/ documentation is delivered to a U.S. Government Agency not within the Department of Defense, then it is delivered with "Restricted Rights", as defined in FAR 52.227-14, Rights in Data - General, including Alternate III (June 1987).

The information in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. Oracle Corporation does not warrant that this document is error free.

Oracle, SQL*Forms, SQL*DBA, SQL*Loader, SQL*Net and SQL*Plus are registered trademarks of Oracle Corporation.

PL/ SQL and Oracle7 are trademarks of Oracle Corporation.

All other products or company names are used for identification purposes only, and may be trademarks of their respective owners.

MAC003038

ORCL00941714

CHAPTER

5

Oracle Web Agent

The Oracle Web Agent uses the PL/ SQL language to integrate the Oracle7 Server and the World Wide Web. It provides transparent access to Oracle7 directly from the World Wide Web (hereafter Web).

The Oracle Web Agent provides an easy-to-use environment for building applications in PL/ SQL. These applications are then stored in the Oracle7 database. A programmer who has chosen PL/ SQL as the language in which to implement a CGI application need only worry about implementing the logic specific to the application itself, because the Oracle Web Agent provides utilities which take care of the repetitive tasks associated with deploying a CGI application.

The HTML output generated by a PL/ SQL procedure is sent from the Oracle7 Server to the Oracle Web Agent. The output is then passed on to the Oracle Web Listener. The Web Listener sends the generated HTML document back to the Web client.

The HTML administrative forms that simplify the task of configuring the Oracle Web Agent are documented in Chapter 7, "Oracle WebServer Administration Utility."

Components of the Oracle Web Agent

The Oracle Web Agent is a CGI program that the Web Listener executes when a request is received for a dynamic document. This program logs into the database and executes stored PL/ SQL procedures that have been specified as part of the URL.

To facilitate the development of PL/ SQL for the Web Agent, a WebServer Developer's Toolkit is provided. This toolkit includes the following packages:

- HTP (hypertext procedures) and HTF (hypertext functions) ease the generation of HTML tags from within the user's PL/ SQL code. HTP and HTF automatically generate the HTML tags retrieved by the Oracle Web Agent.
- OWA_UTIL contains utility functions to enable the programmer to more rapidly build HTML output.

Note: The Web Agent also includes a PL/ SQL package called OWA that contains utility functions called by the Web Agent to do such things as set up the CGI environment variables and extract the output generated by the PL/ SQL code from a PL/ SQL table. A PL/ SQL developer does not make calls to the OWA package directly, although it can be called from a URL and, therefore, directly by a user..

For detailed information about the Oracle WebServer Developer's Toolkit, see Chapter 6. For information on the WebServer Administration Utility for administering the Web Agent, see Chapter 7, "Oracle WebServer Administration Utility."

Who Uses the Oracle Web Agent

The Oracle Web Agent can be used by:

- A programmer who wants to develop Web applications that interact with Oracle7. The Oracle Web Agent enables the programmer to create PL/ SQL procedures that perform database operations and format the results into HTML documents. These documents then get returned to the Web client.

Therefore, the programmer needs to know the following:

- the hypertext functions and hypertext procedures that can be used to generate an HTML-formatted document. These are described in Chapter 6, "The WebServer Developer's Toolkit."

- the basics of the PL/ SQL procedural language. These are contained in Chapter 4, "Introduction to the Oracle7 Server and to PL/ SQL".

In addition, the following is useful for the programmer:

- which Common Gateway Interface environment variables are available to the PL/ SQL application, and how to access them (This is covered in this chapter.)
- how the Oracle Web Agent is invoked (This is covered in this chapter.)
- how the Web Agent handles parameters (covered in this chapter.)
- how to use the OWA_UTIL PL/ SQL Utility Package (covered in Chapter 6, "The WebServer Developer's Toolkit").
- An administrator who wants to install or set up the Oracle Web Agent, and use the Administration Utility to maintain the Web Agent.

The administrator needs to know the following:

- how to create, modify, and delete Web Agent services
- where to install the Developer's Toolkit PL/ SQL packages provided by the Web Agent

The following information is useful for the administrator:

- how the Oracle Web Agent works (covered in this chapter.)
- how the Web Agent handles errors (covered in this chapter.)

How the Oracle Web Agent Works

The Oracle Web Agent enables you to create PL/ SQL stored procedures that can access data in Oracle7, dynamically create an HTML document from that data using the Developer's Toolkit, and return that document to the client.

This is a step by step description of how the Oracle Web Agent works:

1. The user submits a URL from the browser.

The user submits the URL by one of the following methods:

- by selecting a link from an HTML page
- by specifying a URL in the Open URL box of their browser

- by submitting an HTML form

This action invokes a GET or POST request method operation on the associated URL. The request method is passed to the Web Listener using the HTTP protocol.

2. The Web Listener starts the Web Agent.

The Web Listener determines that the URL is associated with the Oracle Web Agent (by extracting the / owa portion of the URL), and spawns the Web Agent. The Web Listener then makes several environment variables available to the Web Agent in compliance with the CGI specification.

3. The Web Agent connects to the Oracle7 Server.

In order to connect to an Oracle7 Server, the Web Agent requires certain information, such as which server to connect to and what username and password to use. This information is stored in the owa. c f g file (located in the OWSAdministrative directory) as part of a Web Agent service. The Web Agent determines which service to use by parsing the SCRIPT_NAME environment variable, which is set up by the Web Listener in accordance with the CGI 1.1 specification. The Web Listener derives the value of this environment variable from the path section just before the / owa in the URL. For example, if / ows-bi n/ hr / owa is the leading portion of the URL, then hr is the service to use.

For information on how to set up a Web Agent service, please see the section, "Web Agent Service", later in this chapter.

4. The PL/SQL procedure is invoked.

After connecting to the database, the Web Agent invokes the appropriate PL/ SQL procedure, whose name is obtained by parsing the PATH_INFO environment variable.

If the PL/ SQL procedure requires parameters, these would have been passed to the Web Agent by the Web Listener through the QUERY_STRING environment variable or on standard input depending on the REQUEST METHOD ("GET" or "POST"). The Web Agent would then pass these on to the PL/ SQL procedure.

For more information on PATH_INFO, SCRIPT_NAME, and QUERY_STRING and how they are used by the Oracle Web Agent, please see the section, "How the Web Agent Uses CGI Environment Variables," later in this chapter.

5. The PL/SQL procedure executes, generating an HTML document.

With the help of the Developer's Toolkit, the PL/ SQL procedure extracts data from the Oracle7 database and generates an HTML document in a PL/ SQL table.

For more information on the Developer's Toolkit, please see Chapter 6, "Oracle WebServer Developer's Toolkit."

6. The Web Agent passes the HTML document produced to the Oracle Web Listener.

If the PL/ SQL procedure terminates without an error, the Web Agent takes the generated HTML document from the PL/ SQL table and sends it to standard output, which is where the Web Listener expects CGI applications to send their output.

If the PL/ SQL procedure returns with an error, the Web Agent sends a programmer-defined error page to standard output.

For more information on how errors are handled, please see the section, "Oracle Web Agent Error Handling".

7. The Web Listener sends the HTML document to the requesting browser.

The Web Listener returns the HTML document to the browser using HTTP. The browser is unaware that the document requested was generated on-the-fly and presents it on the screen like any other HTML document.

Web Agent Service

Since it is desirable that a single Web Listener be able to access multiple databases or schemas within a database, the Oracle Web Agent supports the concept of services. For more information on databases and schemas, see Chapter 4 of this manual. The concept of services is best described by an example:

Company A would like data stored in its Oracle7 database to be accessible to the public via an Oracle Web Listener. However, different departments within Company A have access to different parts of the database (schemas), which are partitioned by way of user names. Thus, the Human Resources department and the Accounts Payable department use different username/ password combinations to log on to the database. If both want to write CGI applications using the Oracle Web Agent to access their data, the Web Agent needs to use the correct username/ password when it logs on to the database. In this scenario, the `owa.cfg` file, which contains the configuration information, will

contain two services, an HR service, and an AP service. Each service will have an associated username/ password, an ORACLE_HOME parameter, and an ORACLE_SID parameter. When a request from a Web browser comes in, the Web Listener will extract the service name that is embedded in the URL and find out which parameters to use by reading the owa.cfg file.

To connect to the Oracle7 Server, the Web Agent needs the following information to be specified in the Web Agent service:

- username
- password
- ORACLE_HOME
- ORACLE_SID (for local databases only)
- SQL*Net V2 Service Name or Connect String (for remote databases only)

The Administration Utility allows the administrator to display, create, modify or delete a Web Agent service. With the Web Agent Creation form, you do not need to modify the configuration file for the Oracle Web Agent service (owa.cfg) directly. See the section "Creating or Modifying Web Agent Services," in Chapter 7 for more information.

The following is an example of a Web Agent service entry in the owa.cfg file, and a corresponding explanation for each parameter:

Developer's T#

```
(
owa_service = es
(
owa_user = www_es
)
(
owa_password = tiger
)
(
oracle_home = /opt/oracle7
)
(
oracle_sid = esprod
)
(
owa_err_page = /es_err.html
)
)
```

```
(
owa_valid_ports = 8000 8888
)
(
owa_log_dir = /opt/oracle7/ows/log
)
(
owa_nls_lang = AMERICAN_AMERICAN.US7ASCII
)
)
```

owa_service

Name of the Web Agent service

owa_user

Database username that the Web Agent will use to connect to the database.

owa_password

Database password that the Web Agent will use to connect to the database.

orade_home

The location of the Oracle7 code tree in the file system. This should be the ORACLE_HOME for the database to which this Web Agent service connects, unless the Web Agent service is set up to connect to a remote database (over SQL*Net). In that case, specify the ORACLE_HOME where the Web Agent is installed. The Web Agent is always installed in an ORACLE_HOME directory or some subdirectory thereof.

orade_sid

Name of the database system ID to connect to. Does not apply for connections to a remote database.

owa_err_page

This is the path of the HTML document that is returned by the Web Agent when an error occurs in the PL/SQL procedure that the Web Agent invoked. This is the actual path, not the virtual path as configured in the Web Listener.

owa_valid_ports

The valid Web Listener network ports the Web Agent will service.

owa_log_dir

The directory where the Oracle Web Agent writes its error file. The error file will have the name `service_name.err`.

`owa_nls_lang`

The NLS_LANG of the Oracle7 database to which the Web Agent connects. If not specified, the Web Agent administration program looks up the database NLS_LANG when the the service is submitted.

How the Web Agent Uses CGI Environment Variables

The Web Agent uses the environment variables shown in Table 5 – 1:

Variable	Contains
REQUEST_METHOD	GET or POST
PATH_INFO	the name of PL/ SQL procedure to invoke
SCRIPT_NAME	contains the service the Web Agent is to use when logging on to Oracle7
QUERY_STRING	parameters to the PL/ SQL procedure (for GET method only. POST method parameters are passed via standard input.)

Table 5 – 1 CGI Variables Used by the Oracle Web Agent

The following is an example of how a typical URL is parsed by the Web Listener in compliance with the CGI 1.1 specification.

`ht t p : // www . nhl . com 8080 / ows - bi n / nhl / owa / hockey . pass ? per son = Gr et zky`

1. The substring `ht t p : // www . nhl . com 8080` in the above URL signals the Web browser to connect to the `www . nhl . com` host's port 8080 using the HTTP protocol.
2. When the Oracle Web Listener that is running on `www.nhl.com` receives the request, the substring `/ ows - bi n / nhl / owa` signals the Web Listener to execute the Web Agent instead of returning a file to the browser as it normally would have done with a static HTML document.
Please see the section on CGI (Common Gateway Interface) in Chapter 3, "The Oracle Web Listener," for more information on how the Web Listener recognizes the difference between a CGI program and a static HTML document.
3. After spawning the Web Agent, the Web Listener passes `/ ows - bi n / nhl / owa` to the Web Agent as the environment variable

SCRIPT_NAME. The rest of the URL is passed to the Web Agent in this manner: The substring / hockey_pass gets passed to the Web Agent in PATH_INFO and the substring "person=Gretzky" is passed to the Web Agent in QUERY_STRING.

4. The Web Agent parses the SCRIPT_NAME to extract the service name nhl .
 - Note that if the Web Agent cannot resolve the passed service name, it defaults to owa_default_service.
5. The Web Agent parses PATH_INFO to determine that hockey_pass is the PL/ SQL procedure to invoke after logging on to Oracle7.
6. The Web Agent makes the parameters passed to it in QUERY_STRING available to the PL/ SQL procedure it invokes.

Note: The fact that the QUERY_STRING environment variable is used indicates that this is a GET request. You can also pass parameters using the POST method. For an example of how to pass parameters using the POST method, please see the next section, "Passing Parameters to PL/ SQL".

Passing parameters to PL/SQL

A PL/ SQL procedure often requires parameters in order to execute and generate the appropriate HTML document. The following section discusses several key concepts and tips that a PL/ SQL developer should understand with respect to how parameters get passed to the specified PL/ SQL routine.

These key concepts and tips are:

- Getting Parameters from the Web Browser to the Web Agent
- Passing Parameters Using an HTML Form
- Ordering Parameters
- Defaulting Parameter Values
- Using Multiple HTML Form Fields with the Same Name
- Overloading Procedures

Getting Parameters from the Web Browser to the Web Agent

Depending on the REQUEST_METHOD used, parameters are passed from the Web Browser to the Web Listener to the Web Agent in one of two ways:

- **QUERY_STRING environment variable** — If the GET method is used by the Web browser, the Web Listener passes the parameters to the Web Agent in this environment variable, as shown in the example in the preceding section.
- **standard input** — If the POST method is used, the Web Listener passes the parameters to the Web Agent using standard input.

The method used to pass parameters from the Web Listener to the Web Agent is transparent to the PL/ SQL procedure that is the actual consumer of the parameter(s). This is an important feature of the Oracle Web Agent: the PL/ SQL programmer need not be aware of which method is used and need not be concerned with parsing either the QUERY_STRING environment variable or standard input. Thus, the PL/ SQL programmer can concentrate on what he or she knows best: developing the logic to extract data from the Oracle database, based on pre-parsed parameters passed by the Oracle Web Agent.

It is recommended that you use POST whenever possible. GET is the method used for links and non-form URLs. For HTML forms, one has a choice. Because the GET method uses operating system environment variables, there are limits on the length of the QUERY_STRING.

Passing Parameters Using an HTML Form

The following example is analogous to the one in the previous section, except that it uses an HTML form that employs the POST REQUEST_METHOD.

```
<FORM METHOD="POST"
ACTION="http://www.nhl.com/8080/ows-bin/nhl/owa/hockey.pass">
Please type the name of the person you wish to search for:
<INPUT TYPE="text" NAME="person"><P>
To submit the query, press this button:
<INPUT TYPE="submit" VALUE="Submit Query"> <P>
</FORM>
```

The above form will cause the Oracle Web Listener to behave in the same fashion as the previous example, except that instead of populating the QUERY_STRING environment variable with "person=Gretzky," the Web Listener will write "person=Gretzky" to standard input. This assumes, of course, user typed "Gretzky" in the entry field of the HTML form above.

Note that the name of the HTML input variable, in this case "person", has to be the same as the PL/ SQL parameter it is to match.

The PL/ SQL procedure that is the recipient of the above parameters follows:

```

create or replace
procedure hockey_pass (person in varchar2) is
    n_assists integer;
begin
    select num_assists into n_assists
    from hockey_stats
    where name=person;
    ht p.print (person||' has '||to_char(n_assists)||' assists this
    season');
end;

```

Ordering Parameters

Generally, the PL/ SQL developer does not need to be concerned with the order in which the Oracle Web Agent receives parameters from an HTML form or through a URL. The only case that it will be relevant is when passing multiple values for the same form field. See "Using Multiple HTML Form Fields with the Same Name" later in this section.

Defaulting Parameter Values

If the PL/ SQL developer cannot guarantee that a value will be passed from a Web Browser for a particular PL/ SQL procedure parameter, then it is recommended that he or she give the parameter a default value. For example:

```

create or replace procedure showvals(a in varchar2 DEFAULT NULL,
                                     b in varchar2 DEFAULT NULL)
is
begin
    ht p.print ('a = '||a||ht p.br);
    ht p.print ('b = '||b||ht p.br);
end;

```

If the Web Agent were to receive a request to call procedure showvals where there was no value for "a", the value for "b" was, say, "Hello", and the DEFAULT NULL clause was not part of the procedure's definition, then the request would generate an error with the following message:

```

OWS-05111: Agent : no procedure matches this call
OWA SERVICE: test_service
PROCEDURE: showvals
PARAMETERS:
=====
B:
Hello

```

By "defaulting" the parameters, the above request would properly output:

a =

b = Hello

which to the end user would look like:

a =

b = Hello

Using Multiple HTML Form Fields with the Same Name

There are a number of instances where one would want to have multiple values passed for the same HTML form variable, and hence to the same PL/SQL parameter. To handle this situation, one can use PL/SQL tables to create an array of values.

One case where multiple values are passed with the same HTML form variable name is in the use of the HTML form tag "SELECT". If one sets the SIZE parameter to something greater than 1, then the user will be able to select multiple values for the same form field.

Another case where one has a set of values corresponding to a single form field follows:

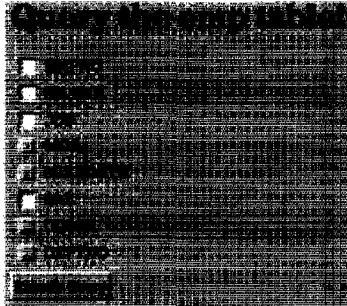
— QUERY_FORM prints an HTML page with all the columns for the
— specified table. Invoke the procedure from a Web Browser with
— a URL like:
http://yourhost:port_num/service_name/owa/query_form?the_table=emp

```
create or replace procedure query_form(the_table in varchar2) is
  cursor cols is
  select column_name
    from user_tab_columns
   where table_name = upper(the_table);
begin
  http.htmlOpen;
  http.headOpen;
  http.title('Query the '||the_table||' table!');
  http.headClose;
  http.bodyOpen;
  — Use owa_util.get_owa_service_path to automatically retrieve
  http.formOpen(owa_util.get_owa_service_path||'do_query');
  — Put in the table as a hidden field to pass on to do_query
  http.formHidden('the_table', the_table);
  — Put in a dummy value, as we cannot DEFAULT NULL a PL/SQL
```

```

table.
httpFormHidden('COLS', 'dummy');
for crec in cols loop
    — Create a checkbox for each column. The form field name
    — will be COLS and the value will be the given column name.
    — Will need to use a PL/SQL table to retrieve a set of
    — values like this. Can use the owa_util.ident_arr type
    — since the columns are identifiers.
    httpFormCheckbox('COLS', crec.column_name);
    httpPrint(crec.column_name);
    http.NL;
end loop;
— Pass a NULL field name for the Submit field; that way, a
— name/value pair is not sent in. Wouldn't want to do this
— if there were multiple submit buttons.
httpFormSubmit(NULL, 'Execute Query');
httpFormClose;
http.bodyClose;
http.htmlClose;
end;
```

Invoking this procedure will bring up a page which looks like:



In this example, the user has already selected to query the EMPNO, ENAME, JOB, and SAL columns:

Here is a procedure to process this form submission:

```

— DO_QUERY executes the query on the specified columns and
— tables. The OWA_UTIL.IDENT_ARR datatype is defined as:
— — type ident_arr is table of varchar2(30) indexed by
— — binary_integer
— create or replace procedure do_query(the_table in varchar2,
—                                     cols in owa_util.ident_arr) is

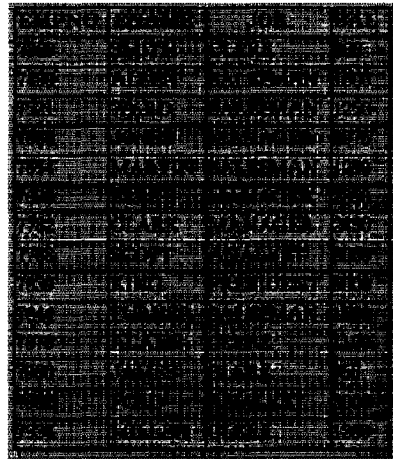
    column_list varchar2(32000);
    col_counter integer;
```

```

    ignore      boolean;
begin
  -- For PL/SQL tables, have to just loop through until you hit
  -- no_data_found. Start the counter at 2 since we put in
  -- a dummy hidden field.
  col_counter := 2;
  loop
    -- build a comma-delimited list of columns
    column_list := column_list || cols(col_counter) || ',';
    col_counter := col_counter + 1;
  end loop;
exception
  when no_data_found
  then
    -- strip out the last trailing comma
    column_list := substr(column_list, 1, length(column_list)-1);

    -- print the table - assumes HTML table support
    ignore := owa_util.tablePrint(the_table, 'BORDER',
                                OWA_UTIL.HTML_TABLE,
                                column_list);
end;
```

Then after selecting the "Execute Query" button, the user would see:



It is a good idea to use a hidden place-holder variable as the first value if you cannot guarantee that at least one value will be submitted for the PL/SQL table. The reason is that one cannot DEFAULT a PL/SQL table. And a call to this procedure with just one argument (the_table) would cause the Web Agent to generate an error.

Note that the Web Agent can only pass parameters to PL/ SQL tables which have a base type of VARCHAR2. This should not provide a significant limitation, as the PL/ SQL type VARCHAR2 is the largest PL/ SQL datatype with a maximum length of 32767 bytes. The values can then be explicitly converted to NUMBER, DATE, or LONG within a stored procedure (using TO_NUMBER or TO_DATE – no conversion needed for LONGs).

Overloading Procedures

PL/ SQL allows developers to overload procedures and functions that are in PL/ SQL packages (but not standalone functions and procedures. For a discussion of overloading as implemented in PL/ SQL, see "Overloading Subprograms" in Chapter 4 of this manual). For example:

```
create or replace package overload is
    procedure proc1(charval in varchar2);
    procedure proc1(numval in number);
end;

create or replace package body overload is
    procedure proc1(charval in varchar2) is
    begin
        http.print('The character value is '||charval);
    end;

    procedure proc1(numval in number);
        http.print('The number value is '||numval);
    end;
end;
```

This functionality can be utilized by the Web Agent, but with the restriction that procedures that are overloaded on datatypes should not have the same parameter names. For example:

```
create or replace package overload is
    procedure proc1(val in varchar2);
    procedure proc1(val in number);
end;
```

When the Web Agent attempts to determine which procedure to call, it will not be able to distinguish between the two and will generate an error.

This limitation is imposed by the lack of HTML-form datatypes, but should not provide a significant limitation, as the PL/ SQL type VARCHAR2 is the largest PL/ SQL datatype with a maximum length of 32767 bytes. The values can then be explicitly converted to NUMBER,

DATE, or LONG within a stored procedure (using TO_NUMBER or TO_DATE – no conversion needed for LONGs).

Accessing CGI Environment Variables

The Oracle Web Listener conforms to the Common Gateway Interface (CGI) 1.1 specification. Thus, all CGI environment variables that are part of this specification are passed from the Oracle Web Listener to the Web Agent. The Web Agent, in turn, makes all relevant environment variables accessible from within PL/SQL.

The Web Agent makes available the CGI environment variables shown in Table 5–2.

Variable	Variable Meaning
AUTH_TYPE	Method used to validate user
GATEWAY_INTERFACE	The revision of the CGI specification to which the server complies
HTTP_USER_AGENT	The browser the client is using to send the request
PATH_INFO	Extra path information given by the client
PATH_TRANSLATED	Translated version of PATH_INFO provided by server for mapping. This consists of the document root concatenated to the PATH_INFO.
REMOTE_HOST	Hostname making the request if it can be determined
REMOTE_ADDR	IP address of the remote host making the request
REMOTE_USER	Used to authenticate user
REMOTE_IDENT	Set to the remote username retrieved from the server
SERVER_PROTOCOL	Name and revision of the information protocol used in the request
SERVER_SOFTWARE	Name and version of information server software answering the request
SERVER_NAME	The server's hostname, or IP address

SERVER_PORT	Port number on which the server is running
SCRIPT_NAME	Virtual path to the script being executed, used for self-referencing URL

Table 5–2 CGI Environment Variables

These environment variables can be accessed from within PL/ SQL using the `owa_util.get_cgi_env` function, which is documented in Chapter 6, "The Oracle WebServer Developer's Toolkit".

Oracle Web Agent Error Handling

There are two types of errors that are handled by the Oracle Web Agent:

- application errors
- system errors

Application Errors

Application errors are specific to the PL/ SQL application. All applications written by a programmer should have their own exception handling in the PL/ SQL procedure that produces the appropriate output.

Because the Oracle Web Agent does not read the HTML output to determine its content, handled exceptions are transparent. As far as the Web Agent is concerned, if the PL/ SQL code generates HTML output, the operation was successful. The user will see whatever handled exception message is generated by the PL/ SQL procedure.

System Errors

System errors are detected by the Oracle Web Agent itself. These are errors that occur when the Web Agent is unable to launch the PL/ SQL procedure or when a PL/ SQL exception is not handled by the stored procedure, causing the exception to be propagated back to the Web Agent as a system error. This causes a standard HTML error document to be returned to the browser.

For example, if the Oracle Web Agent cannot make a connection to the Oracle7 Server, the PL/ SQL procedure cannot run and a system error occurs. The Web Agent then returns a default error message to the browser from the HTTP Server, or returns a customized HTML error

page (if one was previously configured as part of the Web Agent service using the OWA_ERR_PAGE parameter).

How the Web Agent Knows What Error Page to Generate

The Web Agent checks for the error page field in the OWA service configuration. If a value is found there, the Web Agent looks for the specified page in the file system. If found, that page is returned to the Web client. If the lookup fails, a default error message is returned to the Web client.

Creating Error Pages

The Oracle WebServer allows the administrator to create customized error pages that are returned whenever the Web Agent encounters a system error. The purpose of customized error pages is to tell the user what action to take.

Only one custom error page per service can be returned when the Web Agent encounters a system error.

To create a custom HTML error page, use any text or HTML editor you are comfortable with to write your HTML document.

The following is an example of a user-defined HTML error page for a system error:

```
<HTML>
<HEAD>
<TITLE>ERROR</TITLE>
</HEAD>
<BODY>
<HR>
<H1>Your Request Failed</H1>
<HR>
Bob's Big Bulldozer Company could not process your request. Please
try again later, or call us at 1-800-TRACTOR.
Thanks, Bob
</BODY>
</HTML>
```

Once the customized HTML error page is created, the entry for OWA error page in the Web Agent Service configuration file must reflect where the new HTML error page is stored. Use the Web Service Administration page to insert or change the OWA error page entry.

A full path must be specified for the customized HTML error page. If a path is not specified, the Web Agent will search for it in the same directory in which the Web Agent executable is stored.

See the section, "Creating or Modifying the Web Agent Service," in Chapter 7, "Oracle WebServer Administration Utility," for more information.

Error Log File

The Oracle Web Agent keeps an error log file, defined by the OWA Service Name and the OWA Log Directory value for that service. The file will be named `service_name.err`. An entry is appended to this file whenever the Oracle Web Agent encounters an error. The error log records all relevant information, such as date, error number, and URL.

Remember, there is only one error log for each Web Agent service. However, if the Web Agent cannot resolve a service name, and there is no OWA_DEFAULT_SERVICE configured, then errors will be written to OWA.err in the default log directory.

The following is an example of an error log file, and the corresponding meaning for each line in the file:

```
Wed Jun 28 08:14:27 1995                /* Timestamp */
OWA-05100: Agent : unable to connect due to Oracle error 1017
ORA-01017: invalid username/password; logon denied /* Error stack */
OWA SERVICE: HR                        /* Web Agent Service */
PROCEDURE: hrinfo.emp                  /* PL/SQL procedure called */
PARAMETERS:                           /* parameters used by procedure */
=====
ENAME:
RayBor que
```

Oracle Web Agent Version Number

To find out what version of the Oracle Web Agent is installed on your operating system use the following procedure.

At the command line enter the following: (UNIX only)

```
$ORACLE_HOME/ows/bin/owa -v
```

The command will return the following (your result may vary):

```
Oracle Web Agent Release 1.0.0.0.0
```


Exhibit 30

Understanding SQL*Net[®]

Release 2.2

QA
76.9
.D3
U5290
1995

LE[®]
on Age™

ORCL01806620

Understanding SQL*Net[®]

Release 2.2

Part No. A32090-1

ORACLE[®]

Understanding SQL*Net, 2.2

Part No. A32090-1

Copyright © Oracle Corporation 1992, 1993, 1994, 1995

All rights reserved. Printed in the U.S.A.

Primary Authors: Larry Neumann, Sandy Venning, Laura Ferrer

Contributing Author: Shelly Dimmick Willard

Contributors: Romeo Baldeviso, George Chang, Peter Chu, Carl Dellar, John Graham, Jack Haverty, Mark Hill, Mark Jarvis, Mike Kanaley, Ed Miner, Joe Pistrutto, Jay Rossiter, Cyril Scott, Deb Smith, Dave Stowell, Steve Viavant, Rick Wessman, David Wong, Lelia Yin

This software was not developed for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It is the customer's responsibility to take all appropriate measures to ensure the safe use of such applications if the programs are used for such purposes.

This software/ documentation contains proprietary information of Oracle Corporation; it is provided under a license agreement containing restrictions on use and disclosure and is also protected by copyright law. Reverse engineering of the software is prohibited.

If this software/ documentation is delivered to a U.S. Government Agency of the Department of Defense, then it is delivered with Restricted Rights and the following legend is applicable:

Restricted Rights Legend Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of DFARS 252.227-7013, Rights in Technical Data and Computer Software (October 1988).

Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065.

If this software/ documentation is delivered to a U.S. Government Agency not within the Department of Defense, then it is delivered with "Restricted Rights", as defined in FAR 52.227-14, Rights in Data - General, including Alternate III (June 1987).

The information in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. Oracle Corporation does not warrant that this document is error free.

Oracle, SQL*Net, SQL*Connect, SQL*Forms, SQL*DBA, SQL*Loader, SQL*Menu, and SQL*Plus are registered trademarks of Oracle Corporation.

Oracle MultiProtocol Interchange, Oracle Names, Oracle Network Manager, Secure Network Services, and Oracle7 are trademarks of Oracle Corporation.

All other products or company names are used for identification purposes only, and may be trademarks of their respective owners.

QA
76,9

.D3

U5290
1995

CHAPTER

1

Introduction to SQL*Net

SQL*Net release 2.2 is Oracle Corporation's latest remote data access software. It enables both client/server and server/server communications across any network. With SQL*Net, databases and their applications can reside on different computers and communicate as peer applications. Using SQL*Net release 2.2, you can take advantage of all the features of the Oracle Server.

This chapter provides an overview of SQL*Net version 2.0 and later by discussing:

- database capabilities enabled by SQL*Net version 2
- features that are new to this release of SQL*Net
- the relationship of SQL*Net release 2.2 to other Oracle networking products

Functions of SQL*Net Version 2

SQL*Net version 2 provides the basis for two complementary means of distributed communications over a network: client/server and server/server computing.

Distributed Processing In distributed or cooperative processing, clients and servers interact to resolve a single data transaction, even if the applications and databases are separate logical entities, even on separate physical machines. The transaction is distributed between the locations of the client and servers, and the client and servers must cooperate to complete the transaction.

Distributed processing allows the appropriate resources to be allocated from the appropriate machines on the network. The client, for example, can run on a user-friendly graphical workstation or desktop computer while the server resides on a machine more suited for efficient data processing. Distributed processing also allows data processing to be centralized so that many client applications can concurrently access a single server.

SQL*Net in Distributed Processing SQL*Net is responsible for enabling communications between the cooperating partners in a distributed transaction, either client/server or server/server. Specifically, SQL*Net enables clients and servers to connect to each other, send data such as SQL statements and data responses, initiate interrupts from the client or server, and disconnect when the session is complete. During the life of the connection, SQL*Net resolves all differences between the internal data representations and/or character sets of the computers being used.

When a client or server makes a connection request, SQL*Net receives the request and, if more than one machine is involved, passes the request to its underlying layer, the transparent network substrate (TNS), to be transmitted over the appropriate communications protocol to the appropriate server. On the server, SQL*Net receives the request from TNS and passes it to the database as a network message with one or more parameters (that is, a SQL statement).

With the exception of the initial connection, both the local and remote applications generate the same requests whether they run on the same computer or are distributed across multiple computers. The initial connection differs only in that the name of the remote database must be specified by the application or user.

Figure 1 – 1 shows a local application (on the left), contrasted with a client/server application (on the right). In each case, the application is a SQL*Forms data entry screen.

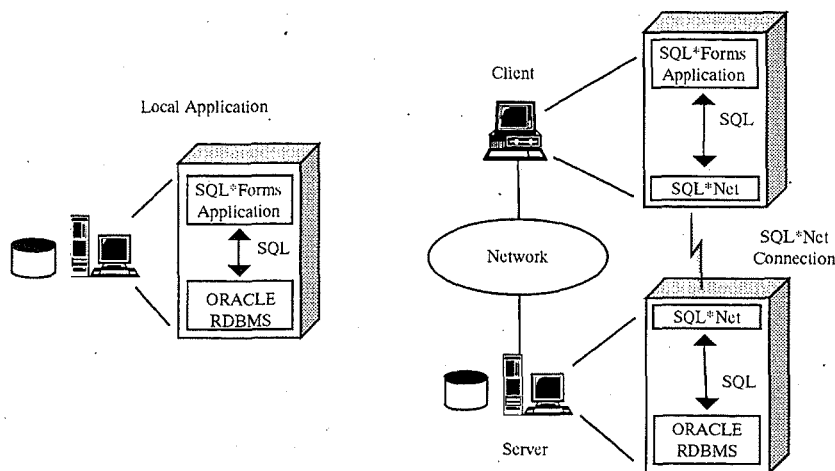


Figure 1 – 1 Local and Distributed Processing

SQL*Net Functional Benefits

SQL*Net V2 provides the following benefits to users of networked applications:

- network transparency
- protocol independence
- media/topology independence
- heterogeneous networking
- location transparency
- good diagnostic capability

Network Transparency

Oracle applications developed with a local database can be distributed across a network to access the same, or a similarly formatted, Oracle database with no changes to the application. SQL*Net is responsible for forwarding application requests for data from an Oracle client or server to a server and returning the results to the initiator of the query. From the application developer's or application user's perspective, all data interaction using SQL*Net is invisible to the user or the application. Additionally, it is possible to change the network structure beneath the application without changing the application. This quality of being invisible is known as *network transparency*.

Protocol Independence SQL*Net provides protocol independence to its applications. An application using SQL*Net can run over any network protocol. Any application built on any computer running any protocol can be distributed without change to other computers running other protocols.

Media/Topology Independence When SQL*Net passes control of a connection to the underlying protocol, all media and/or topologies supported by that network protocol on that platform are indirectly inherited by SQL*Net. SQL*Net allows the network protocol to use any means of data transmission, such as Ethernet, Token Ring, FDDI, or SDLC, to accomplish the low-level data link transmission between the two computers.

Heterogeneous Networking Oracle's client/server and server/server models provide the capability for connectivity across multiple network protocols, each in a manner appropriate to its function.

Client/Server Configuration



Intchg

With SQL*Net Version 2, the client and server can belong to different *communities* connected by one or more MultiProtocol Interchanges. (For information on the MultiProtocol Interchange, see the next section and refer to the *Oracle MultiProtocol Interchange Administrator's Guide*. A community is a group of computers that can communicate using the same transport level protocol, such as TCP/IP; that is, computers that share a common protocol are said to be members of the same community. Using an Interchange as an intermediary, applications on the client and server machines can communicate in spite of having no common transport protocol. Any data exchanged in the client/server applications is forwarded through the Interchanges along the path.

Figure 1 – 2 shows a connection between a client and a server running different protocols in adjacent communities. A MultiProtocol Interchange joins the two networks. SQL*Net and an Oracle Protocol Adapter specific to Protocol A are installed on the client while SQL*Net and an Oracle Protocol Adapter specific to Protocol B are installed on the database server. The Interchange has adapters for both Protocol A and Protocol B.

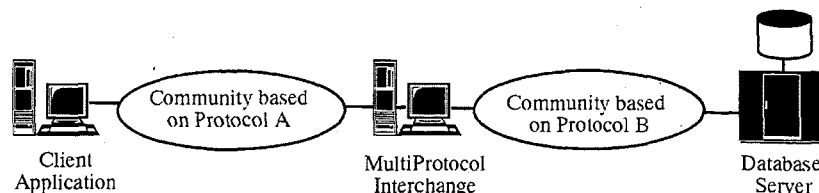


Figure 1 – 2 Heterogeneous Networking with a Client/Server Connection

Server/Server Configuration

In a server/server configuration, this same heterogeneous network capability is extended to include database to database communications for distributed queries and updates in Oracle7. Two types of server/server connections are possible using SQL*Net V2:

- a direct connection between two servers in the same community
- a connection between servers in different communities through one or more Interchanges

The example in Figure 1 – 3 shows both types of connections. In this example, Server 1 is a member of two communities, Community A and Community B. A client application in Community A accesses the database server (Server 1) in the same community. Server 1 determines that the transaction must be distributed further to retrieve data from tables in Server 2 and Server 3. Server 1 initiates a connection to Server 2 in Community B to which Server 1 also belongs. Server 1 also initiates a connection to Server 3 through the Interchange installed between Community B and Community C.

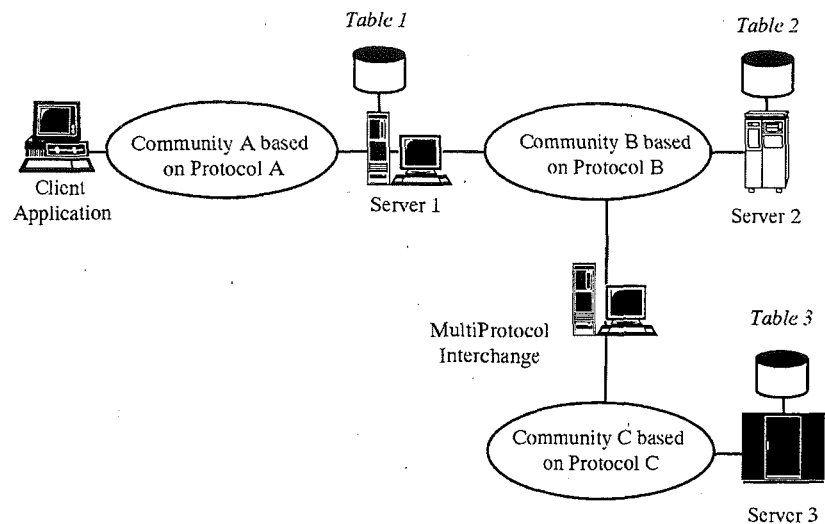


Figure 1 – 3 Heterogeneous Networking in a Distributed Database Transaction

Server 1 does not have to use an Interchange to initiate an additional request for data from Community B since it belongs to both Community A and Community B, but it has to use an Interchange to access a server in Community C.

Note that using the Interchange imposes no new restrictions on a SQL*Net connection. If the MultiProtocol Interchange is used in a client/server connection, clients have a standard peer-to-peer connection to the server despite the fact that they are in different communities. Similarly, if a server initiates a connection with another server through an Interchange using a database link, the standard database link restrictions apply.

Location Transparency

The Oracle server provides the means to make data objects such as tables in remote databases look like they are in the local database to an application developer or user of that data object. The *database link* and *synonym* database features allow the database in which they are created to identify a remote data object and make the location transparent.

This *location transparency* removes all references to the location of the data from applications when the synonym is used. Should the location of the remote table move to another machine or database, only the synonym and database link need to be updated to reference the new location; no applications changes are required. (Database links and synonyms are discussed further in NO TAG“Heading Title”{(on page NO TAG)(in Chapter NO TAG)(in chapter “Chapter Title”)}.Chapter 5,

"Using SQL*Net". Also refer to *Oracle7 Server Distributed Systems, Volume I*.)

Moreover, if the database link connection is specified as a service name (or symbolic name) in the configuration file (TNSNAMES.ORA), the database links accessing the data do not have to be changed if the remote database is moved; the only update required is to the TNSNAMES.ORA file. Similarly, if Oracle Names or an Oracle Native Naming Adapter is used, only the central network definition needs to be changed.



The network definition, TNSNAMES.ORA, and other configuration files are created using Oracle Network Manager. The configuration files are described in Appendix A, "Contents of the SQL*Net Configuration Files." For detailed instructions on creating the network definition and the configuration files for SQL*Net and other Oracle networking products, see the *Oracle Network Manager Administrator's Guide*

SQL*Net Release 2.2 and Other Oracle Network Products

SQL*Net release 2.2 is based on the Oracle Transparent Network Substrate (TNS) technology. It works closely with other TNS-based products to provide easy to administer, flexible, transparent communication between client and server applications. At this time, the following Oracle products are bundled with SQL*Net release 2.2:

- Oracle Network Manager
- Oracle Names
- Oracle Native Naming Adapters
- SNMP Support

The following Oracle products are optional (that is, they are not bundled with SQL*Net release 2.2).

- Oracle Protocol Adapters
- MultiProtocol Interchange
- Secure Network Services
- SQL*Net/DCE

Please refer to Chapter 2 for more information about TNS.

Oracle Network Manager

Oracle Network Manager version 3.0 is a tool that enables you to create the configuration files needed by Oracle networking products. It includes a graphical user interface which enables you to view the network in a hierarchical, or "tree view" structure, so you can see the relationships among the network services, or in a map view which shows the network services as they relate to the user's actual network. Also provided is a context-sensitive online help system and an online walkthrough that takes you through the basic steps of configuring a network.

Oracle Network Manager enables you to validate the network so you can ensure that the resulting configuration files are free of common syntax errors and contain all required information. It includes utilities to assist in converting the configuration files of previous versions of SQL*Net to be compatible with the current version. For further information about Oracle Network Manager, including detailed instructions on how to use it to build a network, consult the *Oracle Network Manager Administrator's Guide*. Oracle Network Manager should be used only by network administrators, not individual users.



NetMan

Oracle Names



ONames

Oracle Names is Oracle Corporation's transparent naming software. Oracle Names provides central storage of network names and addresses so that network components can contact one another easily without regard to their physical locations or specific configurations on the network. Access to the central storage is through Names servers on the network. Oracle Names is described in detail in the *Oracle Names Administrator's Guide*. However, because it works so closely with SQL*Net Version 2.2, it is discussed in this manual as well. You configure Oracle Names servers by using Oracle Network Manager.

Oracle Native Naming Adapters

Oracle Native Naming Adapters enable network administrators to store Oracle service names in a native naming service already in use in their network environment. Services include Network Information Services (NIS), Distributed Computing Environment's Cell Directory Service (DCE's CDS), Banyan's StreetTalk, and Novell's NetWare Directory Services (NDS). Using an Oracle Native Naming Adapter enables you to continue to use familiar utilities to maintain names of Oracle services. Network administrators can configure each client to use Oracle Names,

TNSNAMES.ORA, or one of the native naming adapters such as NIS, to resolve Oracle service names to addresses.

For information on installing and configuring Oracle Native Naming Adapters, refer to your Oracle platform-specific documentation.

SNMP Support

Oracle SNMP support is provided in SQL*Net release 2.2 (and later) for the listener, Oracle7 Server, Oracle Names, and MultiProtocol Interchange. SNMP support allows a database to be remotely monitored by any SNMP-capable management software in a TCP/IP network. SNMP (the Simple Network Management Protocol) is a de-facto standard underlying many popular network management systems such as Hewlett Packard's OpenView, Novell's Network Management System, IBM's NetView/6000, and SunConnect's SunNet Manager. It enables Oracle products such as the SQL*Net network listener, Oracle7 Server, MultiProtocol Interchange, and Oracle Names to be located, identified, and monitored by a management station running at one or more centrally-located nodes.

For information on configuring SNMP support for the listener, Oracle7 Server, Oracle Names, and MultiProtocol Interchange, see the *Oracle Network Manager Administrator's Guide* and to your Oracle platform-specific documentation. Also see Chapter 6, "SNMP Support", in this guide for a general overview of configuring SNMP support for Oracle services. For detailed information about SNMP Support and the information it provides, see the *Oracle SNMP Support Reference Guide*.

Oracle Protocol Adapters

Oracle Protocol Adapters enable SQL*Net to establish connections over specific protocols or networks. Each machine on a network requires at least one protocol adapter, and additional adapters may be required if you use multiple protocols in your network. For example, a machine connecting a TCP/IP network and an SPX/IPX network would require a protocol adapter for each network. Also, the same machine would likely be running a MultiProtocol Interchange to enable communication between the dissimilar networks.

On some platforms, a single Oracle Protocol Adapter can operate on hundreds of different network interface boards, which means that you can deploy applications in any networking environment, including Ethernet, Token-Ring, FDDI (Fiber Distributed Data Interface), ATM (Asynchronous Transfer Mode), wireless and others.

MultiProtocol Interchange



Intchg

The Oracle MultiProtocol Interchange works with SQL*Net Version 2 to provide transparent communication between disparate communities. The product is described in detail in the *MultiProtocol Interchange Administrator's Guide*. However, because it works so closely with SQL*Net Version 2, it is mentioned frequently in this manual as well. The MultiProtocol Interchange is configured using the Oracle Network Manager.

Secure Network Services

Oracle Secure Network Services is an optional product that works with SQL*Net release 2.1.4 and later. It enables SQL*Net and related products to use network data encryption and checksumming so that data cannot be read or altered. It protects data from unauthorized viewing by using the RSA Data Security RC4™ encryption algorithm. To ensure that data has not been modified, deleted, or replayed during transmission, Secure Network Services optionally generates a cryptographically secure message digest and includes it with each packet sent across the network.

Secure Network Services is supported by the MultiProtocol Interchange which means that clients and servers using different protocols can securely transfer data across network protocol boundaries. For example, clients using LAN protocols such as NetWare (SPX/IPX), can securely share data with large servers using different protocols such as LU6.2, TCP/IP, or DECnet.

Release 1.1 provides a standard, highly optimized 56-bit key DES encryption algorithm. For customers outside the USA and Canada, release 1.1 also offers DES40, which combines the standard DES encryption algorithm with the international availability of a 40-bit key. Secure Network Services version 1.0 includes the RC4 encryption algorithm and DES40.

These services are available to most products that sit on top of SQL*Net, including the Oracle7 Server, the CDS tools, and any other Oracle or third-party product that supports SQL*Net.

Future releases of Secure Network Services will include authentication services such as Kerberos, Smart Cards, and biometric authentication methods. These authentication services enhance Oracle7's existing security facilities such as access control logon, roles, and auditing by providing reliable user identification. No changes to applications are required, and SNS works over all protocols, operating systems, or name services.

SQL*Net/DCE

SQL*Net/DCE is an optional product that works with Oracle 7.1.6 (or later) and SQL*Net 2.1.6 (or later). It enables users to transparently use Oracle tools and applications to access Oracle7 servers in a DCE environment. It provides authenticated RPC (Remote Procedure Call) as the transport mechanism, which enables multi-vendor interoperability. The DCE Security service enables a user logged onto DCE to securely access any Oracle application without having to specify a username or password. This is sometimes referred to as "external authentication," formerly referred to as "OPS\$ support".

SQL*Net/DCE also provides DCE Cell Directory Service (CDS), which allows Oracle7 services to be transparently accessed throughout the DCE environment. Users can connect to Oracle database servers in a DCE environment using familiar Oracle service names. Oracle service names can be managed from a central location with standard DCE tools. For information on configuring SQL*Net/DCE see the *SQL*Net/DCE Administrator's Guide* and your Oracle platform-specific documentation.

SQL*Net's New Features

This version of SQL*Net provides several enhancements over previous versions.

New Since SQL*Net Version 1

- Support for the MultiProtocol Interchange

The MultiProtocol Interchange and SQL*Net allow communication between applications over different protocols. For example, a client in an SPX/IPX community can communicate with a server in a TCP/IP community.

- Support for the Oracle7 multi-threaded server

SQL*Net provides support for the Oracle multi-threaded server. This feature of the Oracle7 server enables multiple calls to access the server using a shared dispatcher. For more information about the multi-threaded server, see *Oracle7 Server Concepts*, the *Oracle7 Server Administrator's Guide* and *Oracle7 Server Distributed Systems, Volume I*.

- Improved diagnostic capabilities

SQL*Net version 2 provides tracing and logging abilities that let you record the behavior of each network component and to troubleshoot problems when they occur. For detailed information about tracing and logging, see Chapter 2 in the *Oracle Network Products Messages Manual*.

New Since SQL*Net Version 2.0

- Use with Oracle Names

As described in the preceding section, Oracle Names provides central storage of network names and addresses so that network components can contact one another easily without regard to their physical locations on the network. Oracle Names also enables central storage of global database links created automatically in Network Manager for each database defined. Although SQL*Net does not require Oracle Names to provide transparent access across a network (TNSNAMES.ORA can be used), Oracle Names makes administration and maintenance of a SQL*Net network easier.

- GUI-based Network Manager tool

Administrators create and modify the configuration files required by the Oracle networking products by using the Oracle Network Manager. This tool provides a graphical user interface and extensive validation features to make configuring the network relatively simple. Not only is using the Network Manager faster, easier, and more accurate than creating the files by hand, but Oracle will only support files that can be created using the Network Manager (except for SQLNET.ORA, which contains parameters that can be included only through manual editing; and PROTOCOL.ORA which must be created manually.)

- Enhanced migration utilities

Moving from SQL*Net version 1 to version 2.0 is made easier by Oracle Network Manager. Oracle Network Manager can import a file of version 1 connect strings and aliases into a TNSNAMES.ORA file (or Oracle Names or a Native Naming Service such as NIS) so that users can use them to connect to network destinations after SQL*Net version 2 is installed. Oracle Network Manager also provides an easy way to create a service name for a SQL*Net version 1 connect string that does not already have an alias mapped to it.

Oracle Network Manager also includes a utility called NETCONV which converts a SQL*Net version 2.0 network definition created

by the version 2.0 configuration tool into a release 2.1 network definition that can be stored and edited by the Network Manager. Network definitions created in Network Manager release 2.1 are automatically converted when you open them in Network Manager version 3.0.

- Use of encrypted passwords

The Listener Control Utility and the Names Server provide for optional passwords to allow only authorized personnel to perform certain administration tasks. Oracle Network Manager provides the option to encrypt these passwords within the configuration files for enhanced security. These passwords and SQL*Net logon passwords are also encrypted over the network.

- Support for prespawnd processes

Prespawnd dedicated server processes provide faster connections to databases by eliminating the time required to spawn a process for each connection request. See the section "Prestarted Dedicated Server Processes" in Chapter 2 for further information.

- Support for dead connection detection (Keep Alive or Dead Man's Handle)

Dead connection detection is a feature that allows SQL*Net to identify connections that have been left hanging by the abnormal termination of a client. When this feature is activated, SQL*Net sends periodic probes to check for broken connections. When one is found, SQL*Net returns an error and the server process exits. This feature minimizes the waste of resources by connections that are no longer valid. It also automatically forces a database rollback of uncommitted transactions and locks held by the user of the broken connection.

New Since SQL*Net Version 2.1

- Graphical display of network configuration in Network Manager version 3.0

Network Manager version 3.0, which runs on Windows, allows administrators to view the network in a "tree view" or a "map view". The tree view shows the network's hierarchical structure, indicating the relationships among the network services, whereas the map view shows the network in a spatial layout connecting the network services as they relate to the actual user's network. Also provided is point and click and drag and drop capability to

make creating network objects quick and easy. A context-sensitive online help system gives you quick information when you need it. An online walk-through takes you step by step through the basic tasks of configuring a network.

- Graphical control tools for listener, Names, and MultiProtocol Interchange

Through Network Manager version 3.0 you now have the ability to use some Listener Control Utility (LSNRCTL), Interchange Control Utility (INTCTL), and Oracle Names Control Utility (NAMECTL) commands. For example, you can now reload a Names server and check its status from Network Manager.

- Direct access to the Server Manager from Network Manager version 3.0

Oracle Server Manager, an administration tool that lets you set startup parameters for the database, can be invoked from within Network Manager version 3.0.

- Added configuration support for new network services

Network Manager version 3.0 lets network administrators configure the enhanced encryption, checksumming, and authentication support included with Secure Network Services.

Configuration support for native naming adapters for Network Information Services (NIS), Distributing Computing Environment's Cell Directory Service (DCE's CDS), Banyan's StreetTalk, and Novell's NetWare Directory Services (NDS) will be provided within the timeframe of SQL*Net release 2.2.

Configuration support for Simple Network Management Protocol (SNMP) is provided. Oracle SNMP Support lets DBAs and network administrators monitor the Oracle environment and query status of network services and Oracle7 databases. For example, it can monitor any number of Oracle7 databases in a network, detect when they start or stop, or monitor current user activity. It can also monitor the Oracle network listener, MultiProtocol Interchange, and Oracle Names for current user activity, and to identify potential problems and bring them to the immediate attention of a DBA or network administrator.

Oracle Gateways can now be configured in Network Manager version 3.0.

- SQL*Net/DCE

SQL*Net/DCE enables Oracle applications and tools to access Oracle7 servers in a DCE environment. It provides authenticated

RPC (Remote Procedure Calls) as the transport mechanism. Also provided is the DCE security service which enables a user logged onto DCE to securely access any Oracle application without having to specify a username or password. Cell Directory Service (CDS) registers Oracle7 services so that users can connect to Oracle servers in a DCE environment using familiar service names.

- Oracle Native Naming Adapters

Oracle Native Naming Adapters let administrators integrate Oracle into their native network environments by storing Oracle service names in native name services such as NIS or DCE's CDS. Native Naming Adapters released with SQL*Net 2.2 include Network Information Services (NIS), Distributed Computing Environment's Cell Directory Service (DCE's CDS). Banyan's StreetTalk and Novell's NetWare Directory Services (NDS) will be released within the timeframe of SQL*Net release 2.2.

- Secure Network Services release 1.1

Secure Network Services enables SQL*Net and products based on it to use network data encryption and checksumming so that data cannot be read or altered. Future releases of Secure Network Services will include network authentication adapters such as Kerberos, smart cards, and retinal scanners which will let users who are logged onto an NOS ("network operating system") or in possession of the appropriate credentials to be able to automatically and securely access any Oracle application without specifying a user name or password. See the *Secure Network Services Administrator's Guide* for more information.

- Client-side load balancing for use with MultiProtocol Interchanges improves client/server performance. When routes to a server through different Interchanges exist, SQL*Net release 2.2 determines for the client which route to the server is the most efficient and establishes the connection accordingly.
- SQL*Net error logging now displays the banners for only those protocols that are loaded and linked into SQL*Net configuration.
- Listener Control Utility STATUS command now shows address on which it is listening.
- Listener Control Utility STATUS command now indicates whether a SQL*Net listener can be queried by any SNMP-based network management system.

- TNSPING Utility enables you to determine whether an Oracle service such as an Oracle database, Oracle Names server, MultiProtocol Interchange or any TNS-based service can be successfully reached. If TNSPING can connect, it displays an estimate of the round trip time (in milliseconds) it takes to reach the Oracle service. If it cannot connect successfully, it displays a message, which lets you see the network error that is occurring without the overhead of a database connection.

CHAPTER

2

SQL*Net Version 2 Architecture

SQL*Net version 2 uses the Transparent Network Substrate (TNS) and industry-standard network protocols to connect a client to a server and establish an Oracle session.

The next few sections provide a discussion of SQL*Net and the role it plays in distributed systems. The following architectural concepts are discussed:

- Transparent Network Substrate
- SQL*Net's communication role
- distributed processing with SQL*Net
- SQL*Net operations
- SQL*Net and the network listener

Transparent Network Substrate (TNS)

Forming the basis for Oracle networking products, the Transparent Network Substrate (TNS) enables Oracle to provide a network of applications above all existing networks of computers. With TNS, peer-to-peer application connectivity is possible where no direct machine-level connectivity exists. *Peer-to-peer* is an architecture in which two or more nodes can communicate with each other directly, without the need for any intermediary devices. In a peer-to-peer system, a node can be both a client and a server.

TNS provides two key features to a TNS-based network product and, in turn, any application built using TNS:

- a single, common interface to all industry-standard protocols
- the ability to connect to applications in physically separate networks through one or more MultiProtocol Interchanges

TNS is the foundation component of all current and planned network products from Oracle. Today, TNS networks connect Oracle clients and servers through SQL*Net version 2. In the future, Oracle Corporation will provide additional TNS-based application connectivity tools.

SQL*Net's Communication Role

Communication between client and server proceeds in a stack-like fashion with corresponding levels communicating in a peer relationship. The logical exchange unit at each layer of the stack conveys the level of generalization employed at that level. The Oracle client and server exchange SQL statements and data rows. At the UPI/OPI (User/Oracle Program Interface) layers, these exchanges translate into series of calls to SQL routines such as logon, parse, execute, and fetch. In a distributed transaction, SQL*Net is responsible for sending information across various networks on behalf of a client application or database server. In such a configuration, there are commonly two types of computers acting as the client and server.

Two-Task Common (on page 2 – 4) ensures that all differences between clients and servers, such as internal datatype representations or NLS character sets, are resolved, allowing the client and server to communicate transparently. SQL*Net relays all communication tasks to TNS through its common entry points. SQL*Net is unaffected by the specific communication mechanism used underneath TNS (for example, TCP/IP, DECnet, shared memory, and so on). The SQL*Net layer handles these calls as a series of Oracle send/receive messages, and TNS in turn processes the packets over the network. The network protocol, not provided by Oracle (typically provided with each particular platform by its vendor), supplies a reliable means of communication between the two tasks.

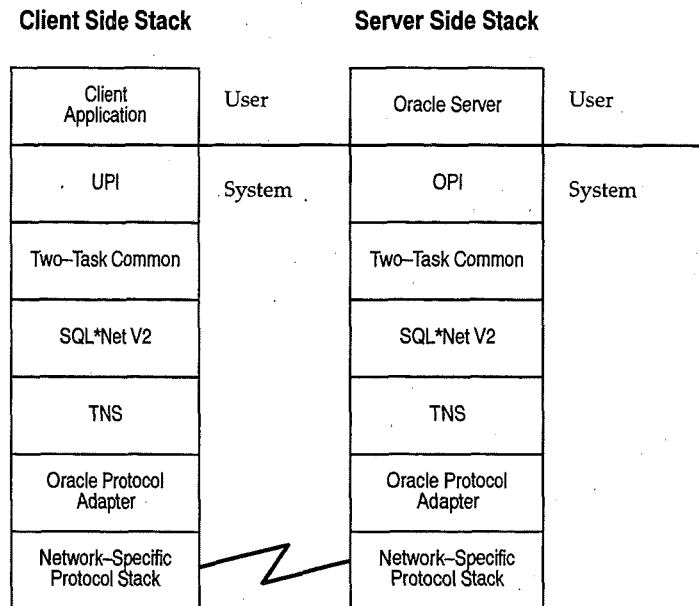


Figure 2 – 1 Oracle Client-Server Components

SQL*Net in Distributed Processing

SQL*Net is responsible for enabling communications between the cooperating partners in a distributed transaction, either client-server or server-to-server. Specifically, SQL*Net enables clients and servers to connect to each other, send data such as SQL statements and data responses, initiate interrupts, and disconnect when the session is complete. During the life of the connection, SQL*Net resolves all differences between the internal data representations and/or character sets of the computers being used.

When a client or server makes a connection request, SQL*Net receives the request. If more than one machine is involved, SQL*Net passes the request to its underlying layer, the transparent network substrate (TNS), to be transmitted over the appropriate communications protocol to the appropriate server. On the server, SQL*Net receives the request from TNS and passes it to the database as a network message with one or more parameters (that is, a SQL statement).

Except for the initial connection, the local and remote applications generate the same requests whether they run on the same computer or are distributed across multiple computers. The initial connection differs only in that the name of the remote database must be specified by the application or user.

Components Involved in Distributed Processing	Several software components are involved in completing a distributed transaction, whether it is a client-server or server-server transaction. Figure 2 – 1 shows the components of a client-server session. These components are described in the following sections.
Client Side Interaction	The following paragraphs discuss the components of the client-server transaction process, beginning with the client application and concluding with the Oracle Server.
Client Application	The client application provides all user-oriented activities, such as character or graphical user display, screen control, data presentation, application flow, and other application specifics. The application identifies any SQL database operations to send to the server database and passes them through the User Program Interface (UPI).
User Program Interface (UPI)	The UPI code contains all information required to initiate a SQL dialogue between the client and the server. It defines calls to the server to: • parse SQL statements for syntax validation • open a cursor for the SQL statement • bind client application variables into the server shared memory • describe the contents of the fields being returned based on the values in the server's data dictionary • execute SQL statements within the cursor memory space • fetch one or more rows of data into the client application • close the cursor The client application uses some combination of these calls to request activity within the server. Often, all UPI calls can be combined into a single message to the server, or they may be processed one at a time through multiple messages to the server, depending on the nature of the client application. Oracle products attempt to minimize the number of messages sent to the server by combining many UPI calls into a single message to the server. When a call is performed, control is passed to SQL*Net to establish the connection or transmit the request to the server.
Two-Task Common	Two-Task Common provides character set and data type conversion between different character sets or formats between client and server. This layer is optimized to perform conversion only when required on a per connection basis.

	At the time of initial connection, SQL*Net version 2 is responsible for evaluating differences in internal data and character set representations and determining whether conversions are required for the two computers to communicate.
SQL*Net	The role of SQL*Net is to establish and maintain a connection between the client application and the server and exchange messages between them. The network listener receives connection requests for a particular database and passes control to the server.
Transparent Network Substrate (TNS)	TNS receives requests from network applications, in this case SQL*Net, and settles all generic machine-level connectivity issues, such as: • the location of the server or destination (open, close functions) • whether one or more MultiProtocol Interchanges will be involved in the connection (open, close functions) • how to handle interrupts between client and server based on the capabilities of each (send, receive functions) The generic set of TNS functions (open, close, send, receive) passes control to an Oracle Protocol Adapter to make a protocol-specific call. Additionally, TNS optionally provides encryption and sequenced cryptographic message digests to protect data in transit. See the <i>Secure Network Services, Release 1.1</i> for more information.
Oracle Protocol Adapter	The Oracle Protocol Adapters are responsible for mapping TNS functionality to industry-standard protocols used in the client-server connection. Each adapter is responsible for mapping the equivalent functions between TNS and a specific protocol.
Network-Specific Protocols	All Oracle software in the client-server connection process requires an existing network protocol stack to make the machine-level connection between the two machines. The network protocol is responsible only for getting the data from the client machine to the server machine, at which point the data is passed to the server-side Oracle Protocol Adapter.
Server-Side Interaction	Going up the process stack on the server side is the reverse of what occurred on the way down the client side. See the right side of Figure 2 – 1. The one operation unique to the server side is the act of receiving the initial connection. The server has a process (the network listener) that regularly checks for incoming connections and evaluates their destination.

The network listener is a process on a server that listens for connection requests for one or more databases on one or more protocols. It is discussed in "SQL*Net and the Network Listener" on page 2 – 10. Based on the Oracle Server ID (SID) specified, the connection is passed to the Oracle Server.

The components above SQL*Net, the OPI and the Oracle Server, are different from those on the client side.

**Oracle Program Interface
(OPI)**

The OPI has a complementary function to that of the UPI. It is responsible for responding to each of the possible messages sent by the UPI. For example, a UPI request to fetch 25 rows would have an OPI response to return the 25 rows once they have been fetched.

Oracle Server

The Oracle Server side of the connection is responsible for receiving dialog requests from the client UPI code and resolving SQL statements on behalf of the client application. Once received, a request is processed and the resulting data is passed to the OPI for responses to be formatted and returned to the client application.

**Server-to-Server
Interaction**

When two servers are communicating to complete a distributed transaction, the process and dialogues are the same as in the client-server scenario, except that there is no client application. See Chapter 5, "Distributed Updates" in *Oracle7 Server Distributed Systems, Volume I* for more information. The server has its own version of UPI, called NPI. The NPI interface can perform all of the functions that the UPI does for clients, allowing a coordinating server to construct SQL requests for additional servers. Figure 2-2 shows a server-to-server connection and all associated layers.

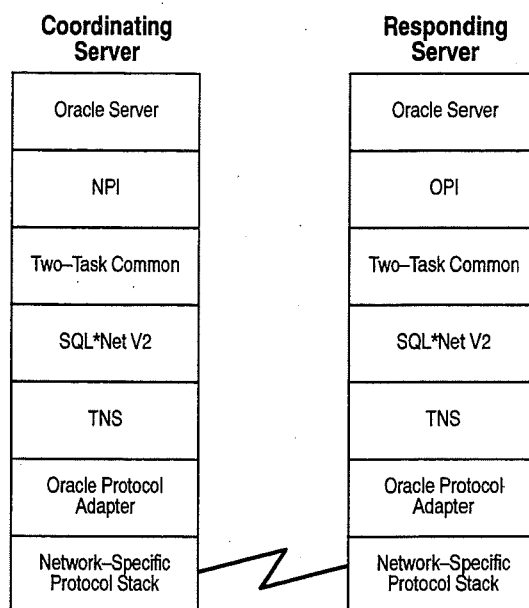


Figure 2 – 2 Oracle Server-Server Components

SQL*Net Operations

SQL*Net provides functions, described in the following sections, that belong to the following classifications:

- connect operations
- data operations
- exception operations

All the functions work with tools and databases that use SQL*Net for distributed processing, although none of them is visible to the user.

Note: The information contained in the following summary is for the benefit of the network administrator, who needs to understand what role SQL*Net version 2 plays within the network.

Connect Operations

SQL*Net supports two basic connect operations:

- connect to server (open)
- disconnect from server (close)

Connecting to Servers

The connect operation is initiated during any standard database login between the client application and the server, with information such as the client machine name and user name being passed to the remote machine. This information is required to support externally-identified logins.

A client application initiates a request for a connection to a remote database (or other network service) by providing a short name for its desired destination. That short name, called a *service name*, is mapped to a network address contained in a *connect descriptor* stored in the network configuration file TNSNAMES.ORA, in a database for use by Oracle Names, or in native naming service such as NIS or DCE's CDS.

Note: If the network includes Oracle Names, the service names and associated connect descriptors are stored in a database that is accessed by the Names servers, and the TNSNAMES.ORA file is not needed. Similarly, if an Oracle Native Naming Adapter such as NIS or DCE's CDS is being used, this information will be stored and retrieved from that native name service.

Disconnecting from Servers

Requests to disconnect from the server can be initiated in the following ways:

- user-initiated disconnect
- additional connection request
- abnormal connection termination
- timer initiated disconnect

User-Initiated Disconnect A user can request a disconnection from the server when a client-server transaction completes. A server can also disconnect from a second server when all server-server data transfers have been completed, and no need for the link remains (the simplest case).

Additional Connection Request If a client application is connected to a server and requires access to another user account on the same server or on another server, most Oracle tools will first disconnect the application from the server to which it is currently connected. Once the disconnection is completed, a connection request to the new user account on the appropriate server is initiated.

Abnormal Connection Termination Occasionally, one of the components below SQL*Net will be disconnected or will abort communications and SQL*Net will not be immediately informed.

During the next SQL*Net data operation, the TNS module will recognize the failure and give SQL*Net a notice to clean up client and server operations, effectively disconnecting the current operation.

Timer Initiated Disconnect or Dead Connection Detection (SQL*Net release 2.1 and later only). Dead connection detection is a feature that allows SQL*Net to identify connections that have been left hanging by the abnormal termination of a client. On a connection with Dead Connection Detection enabled, a small probe packet is sent from server to client at a user-defined interval (usually several minutes). If the connection is invalid (usually due to the client process or machine being unreachable), the connection will be closed when an error is generated by the send operation, and the server process will exit.

This feature minimizes the waste of resources by connections that are no longer valid. It also automatically forces a database rollback of uncommitted transactions and locks held by the user of the broken connection.

Data Operations

SQL*Net supports four sets of client-server data operations:

- send data synchronously
- receive data synchronously
- send data asynchronously
- receive data asynchronously

The concept of sending and receiving data between client and server on behalf of the UPI and OPI is relatively straightforward. A SQL dialogue request is forwarded from the UPI using a send request in SQL*Net. On the server side, SQL*Net processes a receive request and passes the data to the database. The opposite occurs in the return trip from the server.

The basic send and receive requests are synchronous. That is, when the client initiates a request, it waits for the server to respond with the answer. It can then issue an additional request.

SQL*Net version 2 adds the capability to send and receive data requests asynchronously. This capability was added to support the Oracle7 multi-threaded server, which requires asynchronous calls to service incoming requests from multiple clients.

Exception Operations

SQL*Net supports three exception operations:

- initiate a break over the TNS connection
- reset a connection for synchronization after a break
- test the condition of the connection for incoming break

Of these three operations, only the initiation of a break can be controlled by the user. When the user presses the Interrupt key (Ctrl-c on some machines), the application calls this function. Additionally, the database can initiate a break to the client if an abnormal operation occurs, such as during an attempt to load a row of invalid data using SQL*Loader.

The other two exception operations are internal to some products using SQL*Net to resolve network timing issues. SQL*Net can initiate a test of the communication channel, for example, to see if new data has arrived. The reset function is used to resolve abnormal states, such as getting the connection back in synchronization after a break operation has occurred.

SQL*Net and the Network Listener

TNS includes a protocol independent application listener that receives connections on behalf of any TNS application, over any underlying protocol. Referred to as a network listener, it runs as a single process or task and can service the needs of all TNS applications over all protocols available on a machine.

SQL*Net version 2, as a TNS-based product, uses the network listener on a server to receive incoming connections from SQL*Net clients. The network listener listens for SQL*Net connections on a specific port or socket, which is defined in the ADDRESS portion of the connect descriptor.

Network Listener and Native Listeners

The network listener is available for all standard transport protocols supported by TNS. In addition, there are protocols that have application generic listeners or connection acceptance methods, such as DECnet and APPC/LU6.2, that may receive TNS connections.



OS Doc

Additional Information: For information on SQL*Net version 2 connections with a native connection acceptance method, see the Oracle operating system-specific documentation for that protocol and platform.

Prestarted Dedicated Server Processes

SQL*Net release 2.1 and later provides the option of automatically creating dedicated server processes. With this option, when the listener starts, it creates Oracle server processes which are then available to service incoming connection requests. These processes may last for the life of the listener, and they can be reused by subsequent connection requests.

Note: Prespawnd dedicated servers require SQL*Net release 2.1 or later, and require Oracle7 Server release 7.1 or later.

Prestarted dedicated server processes reduce connect time by eliminating the need to create a dedicated server process for each new connection request as it comes to the listener. They also provide better use of allocated memory and system resources by recycling server processes for use by other connections without having to shut down and recreate a server. The use of prestarted dedicated server processes is particularly useful in systems where the Oracle7 multi-threaded server is unsupported, or where the creation of a new server process is slow and resource-intensive.

Figure 2 – 3 shows the role of the network listener in a SQL*Net connection to a server connected to two communities.

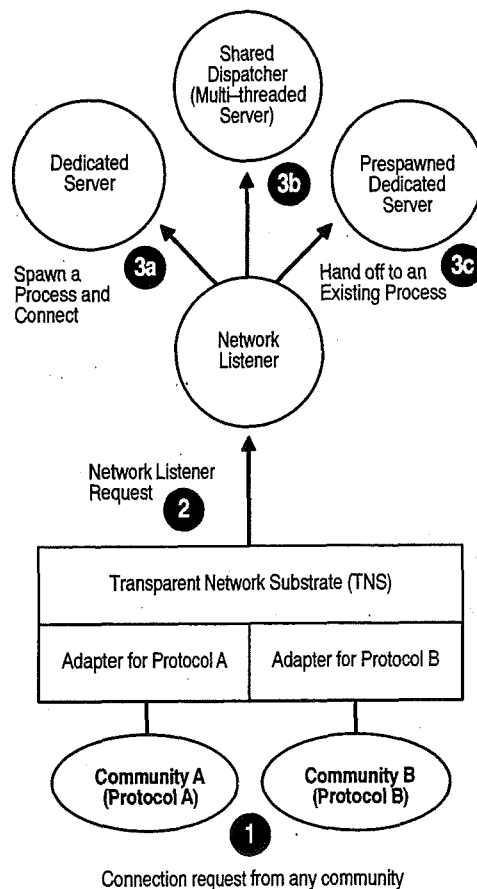


Figure 2 – 3 Network Listener in SQL*Net Connection

The steps involved in establishing a connection (as shown in Figure 2 – 3) are:

- Step 1. A connection request is made by any client in the TNS network and arrives through one of the communities to which the listener is attached.
- Step 2. The network listener identifies that a connection request has arrived in one of its communities.
- Step 3.
 - a. The network listener spawns a dedicated server process and passes control of the incoming connection to it, or,
 - b. the address of a shared dispatcher process (multi-threaded server) is provided, and the incoming connection is directed to it, or,
 - c. the incoming connection is redirected to one of the prespawnd dedicated server processes.

At the completion of a connection, the network listener continues to listen for additional incoming connections.

How SQL*Net Establishes Connections to a Prespawnd Dedicated Server

Prestarted (commonly referred to as "prespawnd") Oracle7 Servers are server processes that are pre-started by the listener before any incoming connection request. They improve the time it takes to establish a connection on servers where the multi-threaded server is not used or not supported on a given machine. Their use in a heavily loaded distributed system can be beneficial.

Set the following parameters using Oracle Network Manager. They control how the server is pre-spawnd.

PRESPAWN_MAX	The maximum number of prespawnd servers the listener creates. This value should be a large number and at least the sum of the POOL_SIZE for each protocol.
--------------	--

POOL_SIZE	The number of unused prespawnd server processes for the listener to maintain on the selected protocol. The number must be greater than zero, but no larger than the PRESPAWN_MAX value. Set this value to the average expected number of connections at any given time.
TIMEOUT	The time that an inactive server process should wait for the next connection before it shuts down. This parameter is used to prevent server processes from being immediately shut down after a client disconnects. For greatest efficiency, provide a short time value for this parameter.

You can set specific prespawnd server parameters for each SID. Thus, systems with heavy use can be tailored to accommodate the larger number of connection requests by setting PRESPAWNED_MAX and POOL_SIZE to large values. Similarly, when systems require mostly shared connections, the number of prestarted servers can be set to a low value.

Following is the sequence of events that occur when you are using prestarted servers to service client connection requests.

1. The listener is started and listens on the addresses pre-configured in LISTENER.ORA, created by the network administrator using Network Manager.
2. The listener then spawns a series of server processes until it reaches the POOL_SIZE for each SID defined in LISTENER.ORA.
3. Each spawned server process performs a wildcard listen and provides the listener with the wildcard address that it is listening on. The listener initially marks all pre-started servers as idle.
4. The client calls the pre-configured well-known address of the listener.
5. The listener receives the connection request, performs the connection handshake and determines if the client is allowed to connect. If not, the listener refuses the connection and then resumes at step 9.
6. The listener issues a redirect message to the client containing one of the wildcard listen addresses of the pre-spawned servers. The listener then logs that server as active.

7. The client dissolves the connection to the listener and then establishes a connection to the pre-spawned server using the address provided in the redirect message.
8. The listener spawned another server to replace the active pre-spawned server (provided the PRESPAWN_MAX value is greater than the number of pre-spawned server processes active and idle).
9. The listener continues listening for incoming connections.

The above sequence of events continues until the PRESPAWN_MAX is reached, at which point the listener will cease spawning new servers.

When clients disconnect, the prespawnd server associated with the client is returned to the idle pool. If then waits the length of time defined in the TIMEOUT parameter to be assigned to another client. If no client is handed to the pre-spawned server before TIMEOUT expires, the pre-spawned server shuts itself down.

See the *Oracle Network Manager Administrator's Guide* for more information on configuring this feature.

How SQL*Net Establishes Connections to a Multi-Threaded Server

The multi-threaded server enables many clients to connect to the same server without the need for dedicated server processes for each client. Using the multi-threaded server enables you to minimize the memory and processing resources needed on the server side as the number of connections to the database increases.

The sequence of events that occurs with the Oracle7 multi-threaded server is as follows:

- The listener and the multi-threaded server start up.
- Clients connect to the Oracle7 multi-threaded server.

What Happens When an MTS and Listener Are Started

During initial startup of the Oracle7 multi-threaded server and the listener, the following sequence occurs:

1. The listener starts and listens for the addresses pre-configured in the LISTENER.ORA file. (The network administrator creates this file with Network Manager.)

2. The DBA starts the Oracle7 database. Dispatchers start according to the configuration parameters in the initialization parameter file. Dispatchers each use one particular protocol. There may be many, on assorted protocols. Each dispatcher performs a wildcard listen for the protocol assigned to it.

Note: A *wildcard listen* is where the server process listens, but informs the underlying protocol stack (or operating system in the case of the IPC Protocol Adapter) that it has no preference as to what address it listens for other than specifying the protocol on which it wishes to perform the operation. As a result, many operating systems will choose a free listening address and automatically assign this to the requesting server process.

3. Each dispatcher calls the listener using the protocol it is assigned, at the address defined in the MTS_LISTEN_ADDRESS in the initialization parameter file.

Note: If step 2 is performed before step 1, the dispatchers will be unable to contact the listener in step 3. If this occurs, each dispatcher loops and attempts to reconnect to the listener every 60 seconds. Meanwhile, incoming connection requests will be handled through other means (prespawned dedicated or newly-spawned dedicated server processes).

The listener and the Oracle7 multi-threaded server should be ready for incoming connections at this point. You can check which dispatchers have registered with the listener by typing

```
lsnrctl services listener_name
```

How a Multi-Threaded ServerConnection Request Is Handled

The following is how a multi-threaded server connection request is handled:

1. The client calls the preconfigured well-known address of the listener.
2. The listener receives the connection request, performs the connection handshake and determines if the client is allowed to connect (by checking the list of SIDs it listens for), at which point it continues with step 6. If not, the listener refuses the connection and then resumes at step 6.
3. The listener issues a redirect message back to the client containing the address of the least-called dispatcher that is listening on the protocol used by the client.
4. The client closes the connection to the listener and then establishes a new connection to the dispatcher, using the address provided by the listener in the redirect message.

5. The listener and dispatcher perform a short handshake to update each other of the presence of a new connection. This is so that the listener can load balance connections between dispatchers running on the same protocol.
6. The listener resumes listening for incoming connections.

When an Oracle7 Server has been configured as a multi-threaded server, incoming connections are always routed to the dispatcher unless the connection request specifically requests a dedicated server (by having `SERVER=DEDICATED` in the `CONNECT_DATA` portion of the connect descriptor) or no dispatchers are available. To create this parameter, create an alias for the database using Network Manager. (An alias is an alternative name that is mapped to a service name.) Use Network Manager to make that alias a dedicated server. See Chapter 5 in the *Oracle Network Manager Administrator's Guide* for more information.

Exhibit 31

**THIS EXHIBIT HAS BEEN
REDACTED IN ITS ENTIRETY**

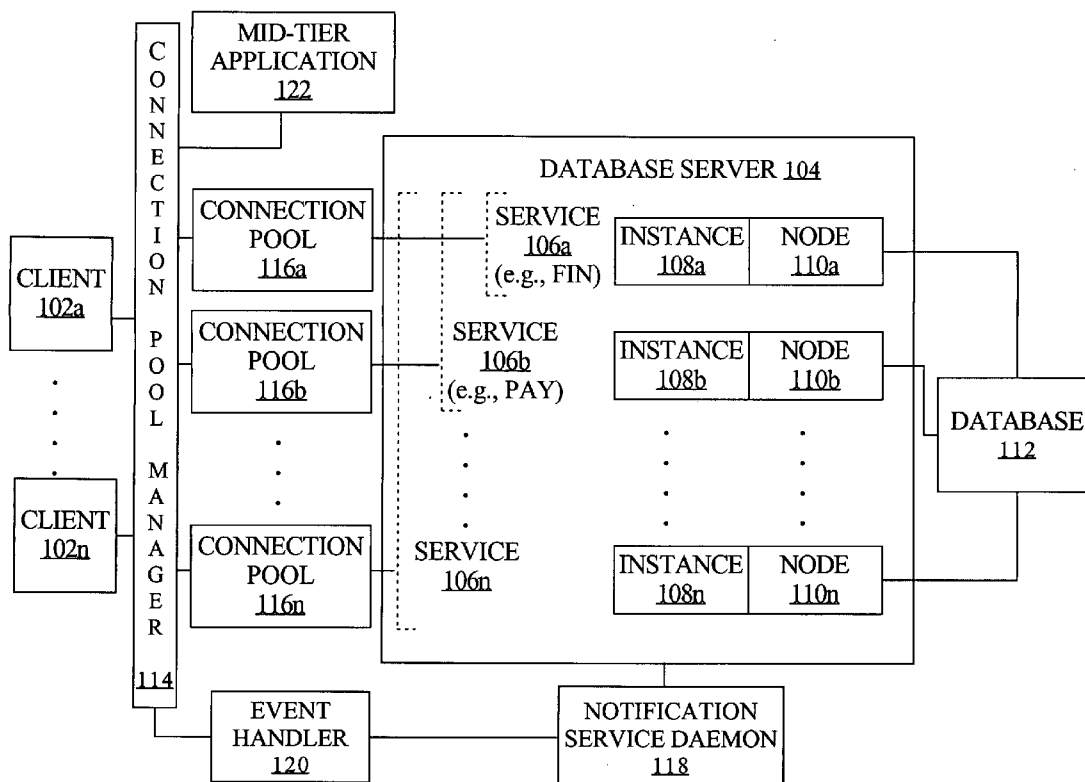
Exhibit 32



US 20050262183A1

(19) **United States**(12) **Patent Application Publication**
Colrain et al.(10) **Pub. No.: US 2005/0262183 A1**(43) **Pub. Date: Nov. 24, 2005**(54) **CONNECTION POOL USE OF RUNTIME
LOAD BALANCING SERVICE
PERFORMANCE ADVISORIES**(75) Inventors: **Carol L. Colrain**, Redwood Shores,
CA (US); **Rajkumar A. Irudayaraj**,
Mountain View, CA (US); **Douglas N.
Surber**, Orinda, CA (US)Correspondence Address:
**HICKMAN PALERMO TRUONG &
BECKER/ORACLE**
2055 GATEWAY PLACE
SUITE 550
SAN JOSE, CA 95110-1089 (US)(73) Assignee: **ORACLE INTERNATIONAL COR-
PORATION**, REDWOOD SHORES,
CA (US)(21) Appl. No.: **11/168,968**(22) Filed: **Jun. 27, 2005****Related U.S. Application Data**(63) Continuation-in-part of application No. 10/917,715,
filed on Aug. 12, 2004.(60) Provisional application No. 60/652,368, filed on Feb.
11, 2005. Provisional application No. 60/500,096,
filed on Sep. 3, 2003. Provisional application No.
60/495,368, filed on Aug. 14, 2003.**Publication Classification**(51) **Int. Cl.⁷** **G06F 17/30**(52) **U.S. Cl.** **709/200**(57) **ABSTRACT**

Runtime connection load balancing of work across connections to a clustered computing system involves the routing of requests for a service, based on the current operational performance of each of the instances that offer the service. A connection is selected from an identified connection pool, to connect to an instance that provides the service for routing a work request. The operational performance of the instances may be represented by performance information that characterizes the response time and/or the throughput of the service that is provided by a particular instance on a respective node of the system, and is relative to other instances that offer the same service.



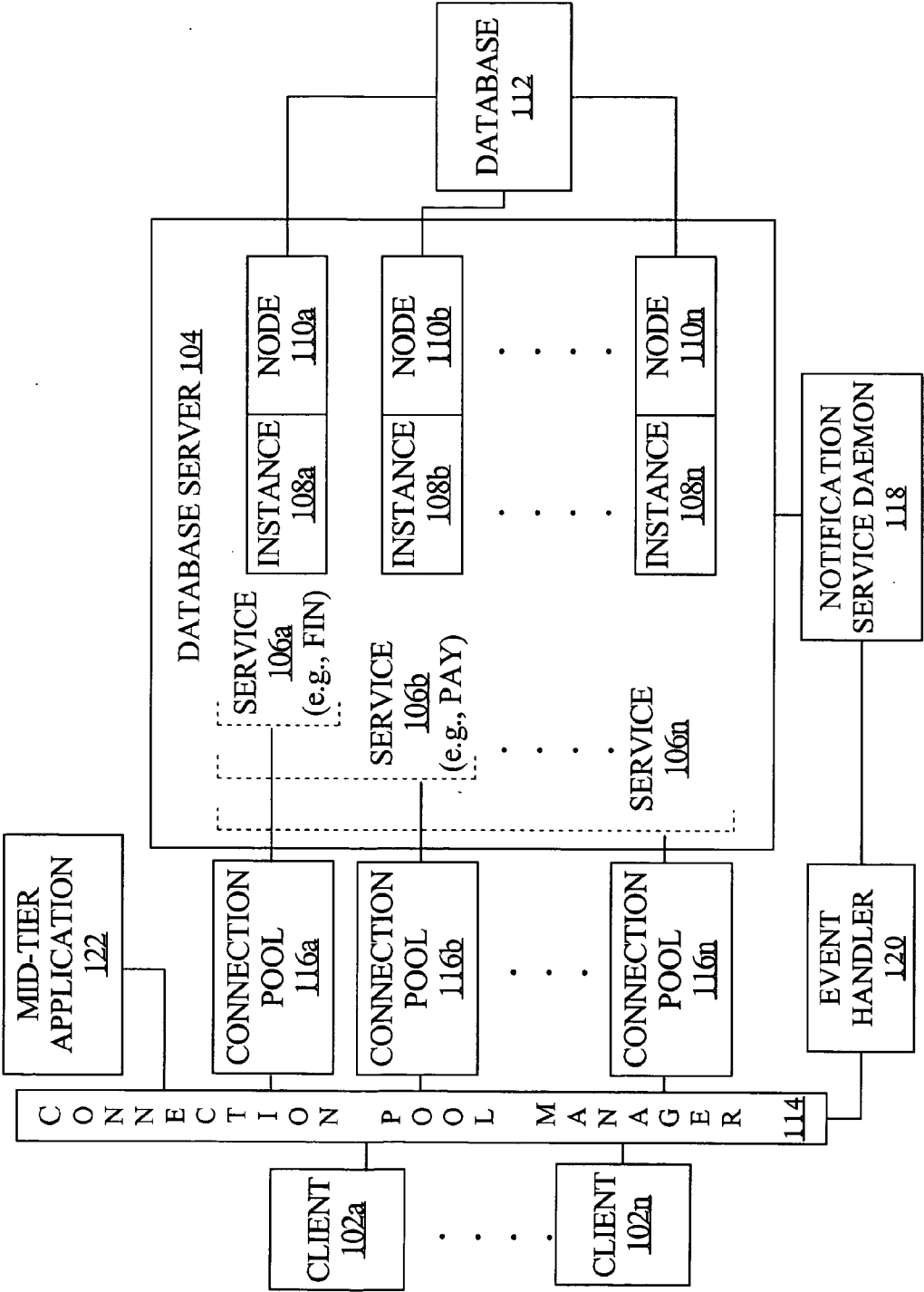


FIG. 1

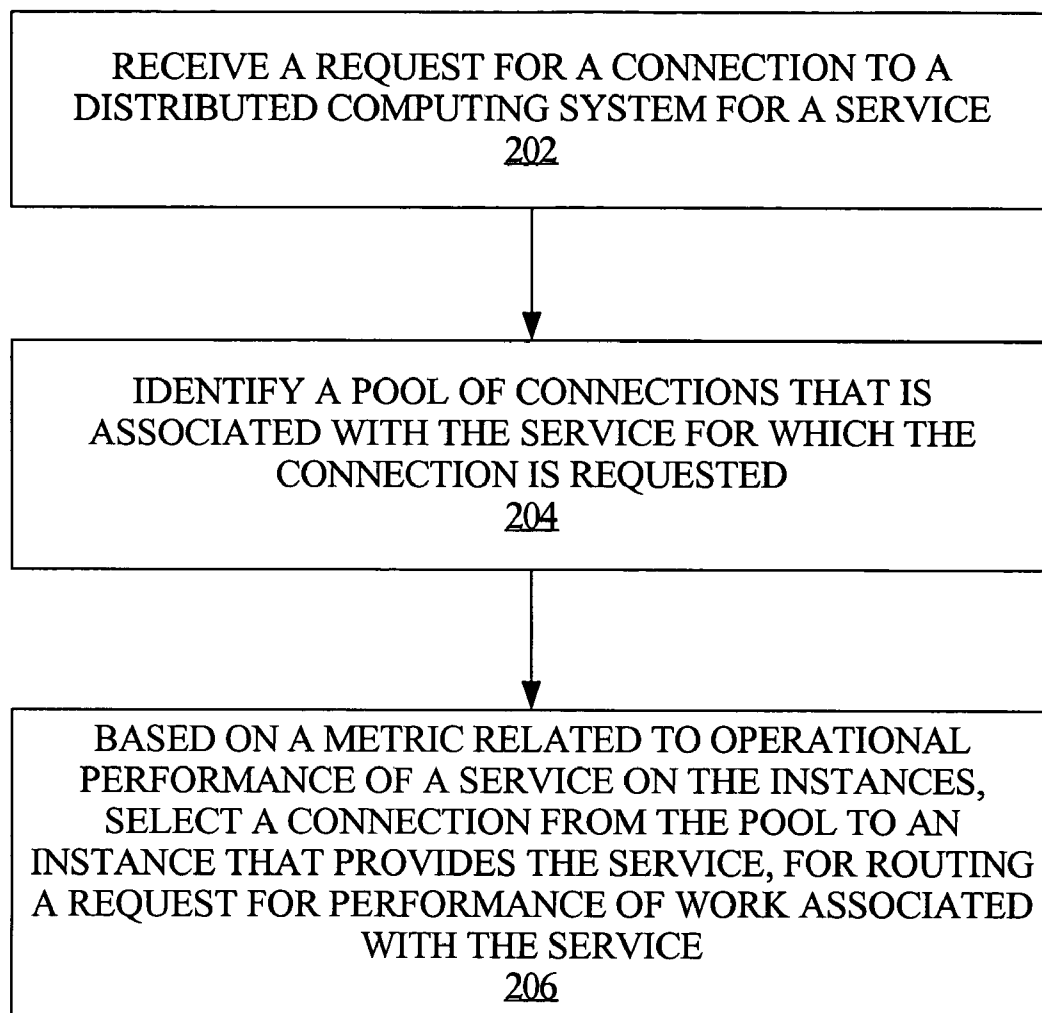


FIG. 2

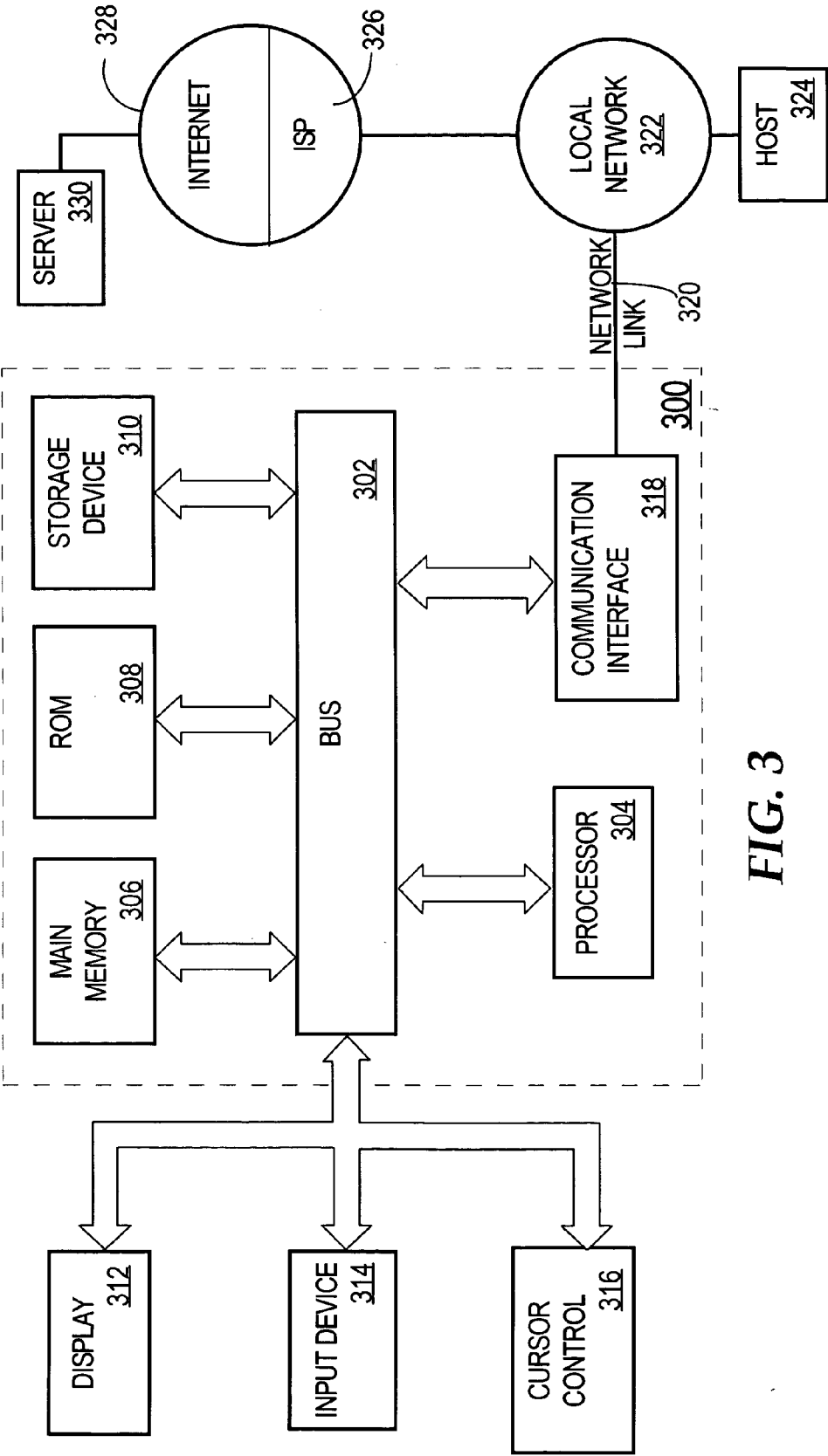


FIG. 3

US 2005/0262183 A1

Nov. 24, 2005

1

CONNECTION POOL USE OF RUNTIME LOAD BALANCING SERVICE PERFORMANCE ADVISORIES

CROSS REFERENCE TO RELATED APPLICATIONS

[0001] This application is related to and claims the benefit of priority to U.S. Provisional Patent Application No. 60/652,368 filed on Feb. 11, 2005, entitled "Runtime Load Balancing Based on Service Level Performance"; the content of all of which is incorporated by this reference in its entirety for all purposes as if fully set forth herein.

[0002] This application is a continuation-in-part of and claims the benefit of priority to U.S. patent application Ser. No. 10/917,715 filed on Aug. 12, 2004, entitled "Managing Workload By Service", which claims the benefit of priority to U.S. Provisional Patent Application No. 60/500,096 filed on Sep. 3, 2003, entitled "Service Based Workload Management and Measurement In a Distributed System" and claims the benefit of priority to U.S. Provisional Patent Application No. 60/495,368 filed on Aug. 14, 2003, entitled "Computer Resource Provisioning"; the contents of all of which are incorporated by this reference in their entirety for all purposes as if fully set forth herein.

[0003] This application is related to and claims the benefit of priority to U.S. Provisional Patent Application No. 60/652,368 filed on Feb. 11, 2005, entitled "Runtime Load Balancing Based on Service Level Performance"; the content of all of which is incorporated by this reference in its entirety for all purposes as if fully set forth herein.

[0004] This application is related to the following applications, the contents of all of which are incorporated by this reference in their entirety for all purposes as if fully set forth herein:

[0005] U.S. patent application Ser. No. 10/917,663 filed on Aug. 12, 2004, entitled "Fast Reorganization Of Connections In Response To An Event In A Clustered Computing System";

[0006] U.S. application Ser. No. 10/917,661 filed on Aug. 12, 2004, entitled "Calculation of Service Performance Grades in a Multi-Node Environment That Hosts the Services";

[0007] U.S. Patent Application No. _____ (Docket No. 50277-2336), entitled "Runtime Load Balancing Of Work Across A Clustered Computing System Using Current Service Performance Levels".

FIELD OF THE INVENTION

[0008] The present invention relates generally to distributed computing systems and, more specifically, to techniques for connection pool use of runtime load balancing service performance advisories.

BACKGROUND OF THE INVENTION

[0009] Many enterprise data processing systems rely on distributed database servers to store and manage data. Such enterprise data processing systems typically follow a multi-tier model that has a distributed database server in the first

tier, one or more computers in the middle tier linked to the database server via a network, and one or more clients in the outer tier.

[0010] Clustered Computing System

[0011] A clustered computing system is a collection of interconnected computing elements that provide processing to a set of client applications. Each of the computing elements is referred to as a node. A node may be a computer interconnected to other computers, or a server blade interconnected to other server blades in a grid. A group of nodes in a clustered computing system that have shared access to storage (e.g., have shared disk access to a set of disk drives or non-volatile storage) and that are connected via interconnects is referred to herein as a work cluster.

[0012] A clustered computing system is used to host clustered servers. A server is combination of integrated software components and an allocation of computational resources, such as memory, a node, and processes on the node for executing the integrated software components on a processor, where the combination of the software and computational resources are dedicated to providing a particular type of function on behalf of clients of the server. An example of a server is a database server. Among other functions of database management, a database server governs and facilitates access to a particular database, processing requests by clients to access the database.

[0013] Resources from multiple nodes in a clustered computing system can be allocated to running a server's software. Each allocation of the resources of a particular node for the server is referred to herein as a "server instance" or instance. A database server can be clustered, where the server instances may be collectively referred to as a cluster. Each instance of a database server facilitates access to the same database, in which the integrity of the data is managed by a global lock manager.

[0014] Services for Managing Applications According to Service Levels

[0015] Services are a feature for database workload management that divide the universe of work executing in the database, to manage work according to service levels. Resources are allocated to a service according to service levels and priority. Services are measured and managed to efficiently deliver the resource capacity on demand. High availability service levels use the reliability of redundant parts of the cluster.

[0016] Services are a logical abstraction for managing workloads. Services can be used to divide work executing in a database cluster into mutually disjoint classes. Each service can represent a logical business function, e.g., a workload, with common attributes, service level thresholds, and priorities. The grouping of services is based on attributes of the work that might include the application function to be invoked, the priority of execution for the application function, the job class to be managed, or the data range used in the application function of a job class. For example, an electronic-business suite may define a service for each responsibility, such as general ledger, accounts receivable, order entry, and so on. Services provide a single system image to manage competing applications, and the services allow each workload to be managed in isolation and as a

US 2005/0262183 A1

Nov. 24, 2005

2

unit. A service can span multiple server instances in a cluster or multiple clusters in a grid, and a single server instance can support multiple services.

[0017] Middle tier and client/server applications can use a service by, for example, by specifying the service as part of the connection. For example, application server data sources can be set to route to a service. In addition, server-side work sets the service name as part of the workload definition. For example, the service that a job class uses is defined when the job class is created, and during execution, jobs are assigned to job classes and job classes run within services.

[0018] Database Sessions

[0019] In order for a client to interact with a database server on a database cluster, a session is established for the client. Each session belongs to one service. A session, such as a database session, is a particular connection established for a client to a server, such as a database instance, through which the client issues a series of requests (e.g., requests for execution of database statements). For each database session established on a database instance, session state data is maintained that reflects the current state of a database session. Such information contains, for example, the identity of the client for which the session is established, the service used by the client, and temporary variable values generated by processes executing software within the database session. Each session may each have its own database process or may share database processes, with the latter referred to as multiplexing.

[0020] Connections are mechanisms for enabling sessions. A connection may refer to either a physical mechanism or a logical connection mechanism, or both. Often, there is a one-to-one mapping of logical connections to physical connections; however, it is possible to have more than one logical connection associated with single physical connection or more than one physical connection associated with a single logical connection. Regardless, management of connections to a clustered database that comprises multiple server instances executing on multiple machines is amplified as the number of instances, machines and clusters increases. An important and challenging feature that is expected from a clustered database is management of many connections to the clustered database server.

[0021] Generally, a connection is a vehicle to perform work with a database. A business transaction may comprise one or more work requests. When using a connection pool, an application "borrows" a connection from a connection pool, executes the work requests, and then returns the connection back into the connection pool.

BRIEF DESCRIPTION OF THE DRAWINGS

[0022] Embodiments of the present invention are depicted by way of example, and not by way of limitation, in the figures of the accompanying drawings and in which like reference numerals refer to similar elements and in which:

[0023] FIG. 1 is a block diagram that illustrates an operating environment in which an embodiment can be implemented;

[0024] FIG. 2 is a flow diagram that illustrates a method for routing work requests to a clustered computing system, according to an embodiment; and

[0025] FIG. 3 is a block diagram that depicts a computer system upon which an embodiment of the invention may be implemented.

DETAILED DESCRIPTION OF EMBODIMENTS OF THE INVENTION

[0026] Techniques for connection pool use of runtime load balancing service performance advisories are described. As described in U.S. patent application Ser. No. _____ (Docket No. 50277-2336), entitled "Runtime Load Balancing Of Work Across A Clustered Computing System Using Current Service Performance Levels", current service level performance information and work distribution advisories are computed and made available to client subscribers, with which work routing decisions can be made. Service level performance information and work distribution advisories regarding the different instances of the system can be used to allow balancing of the work across the system.

[0027] Functional Overview of Embodiments

[0028] Runtime load balancing of work across a clustered computing system involves servers calculating, and clients utilizing, the current service performance levels of each instance in the system. Such performance levels ("performance grades") are based on performance metrics, and corresponding percentage distribution advisories are posted for use by various types of client subscribers.

[0029] Within a multi-instance server, various performance metrics are gathered for each instance. These performance metrics may include operations completed per second, elapsed time per operation, CPU utilization, I/O utilization, network utilization, and the like. A moving average of the metrics is usually used in order to smooth out any short-term variations. In one embodiment, each instance within the server periodically sends its performance metrics to a centralized location.

[0030] The server then computes a performance grade for each instance. The computation used may vary by policy. Examples of possible policies include: (a) using estimated bandwidth as a performance grade, (b) using spare capacity as a performance grade, or (c) using response time as a performance grade. The server may compute a performance grade for each instance without regard to the performance of other instances, or the server may holistically look at all instances to produce a grade for each instance. The server publishes the performance grades of the instances to the client subscribers.

[0031] Using the techniques described herein, clients distribute work requests across servers in a clustered computing environment as the requests arrive. Automatically and intelligently directing work requests to the best server instances, based on real-time service performance metrics, minimizes the need to manually relocate work within the clustered system. In general, basing work request routing decisions on service response time and/or service throughput recognizes, for non-limiting examples, differences in various machine's current workload and computing power, sessions that are blocked in wait mode, failures that block processing, and competing services having different levels of priority.

[0032] Techniques for runtime connection load balancing across a clustered computing system using service performance advisories are described. Such techniques are used to

US 2005/0262183 A1

Nov. 24, 2005

3

route requests for performance of work, based on performance metrics associated with various server instances that offer services that perform the work. Work requests are routed to connections that best serve the work, and such routing techniques respond quickly to changing conditions in the system.

[0033] In one aspect, a request for a connection to a clustered computing system for performance of some work is received, such as by a connection pool manager. A pool of connections to the system is identified, where the pool is associated with a service that performs the work for which the connection is requested. Instances that offer the service that performs the requested work are identified. Based on operational performance metrics associated with the service provided on the various identified instances, a connection is selected from the identified pool, to connect to an instance that provides the service for routing a request for performance of the work.

[0034] Operating Environment

[0035] FIG. 1 is a block diagram that illustrates an operating environment in which an embodiment can be implemented. Embodiments of the invention may be implemented using a connection pool manager that is associated with a clustered computing system, such as a clustered database server comprising multiple database server instances executing on multiple host nodes to access and manipulate shared data stored on a data storage mechanism. The example operating environment of FIG. 1 is illustrated and described in the context of a clustered database server for purposes of discussion, however, it is contemplated that any distributed database server or distributed computing system could benefit from use of the broader techniques enabled herein. Therefore, embodiments of the invention are not limited to use with a clustered database system.

[0036] One or more clients 102a-102n are communicatively coupled to a distributed, or clustered, database server ("database server") 104. Database server 104 refers to database server instances 108a-108n and nodes 110a-110n on which the instances execute. Other components may also be considered as part of the database server 104, such as a connection pool manager 114, connection pools 116a-116n, a notification service daemon 118 and an event handler 120. The actual architecture in which the foregoing components are configured may vary from implementation to implementation.

[0037] In FIG. 1, connection pool manager 114 and connection pools 116a-116n are depicted as components not within database server 104 for purposes of example. Connection pool manager 114 is communicatively coupled to and utilized by an application server or, generally, a middle tier application 122. In addition, connection pool manager 114 is logically coupled to connection pools 116a-116n, which are logically coupled to database server 104. Notification service daemon 118 and event handler 120 are also depicted as not within database server 104, and are communicatively coupled to database server 104, in particular embodiments.

[0038] Clients 102a-102n are applications that execute processes on the database server 104 via, for example, a network. One server instance may be a client of another server instance.

[0039] Database 112 comprises data and metadata that is stored on a persistent memory mechanism, such as a set of hard disks that are communicatively coupled to nodes 110a-110n, each of which is able to host one or more instances 108a-108n, each of which hosts at least a portion of one or more services. Such data and metadata may be stored in database 112 logically, for example, according to object-oriented constructs, object-relational constructs, relational constructs, multidimensional constructs, or a combination of relational and multidimensional database constructs. Nodes 110a-110n can be implemented as a conventional computer system, such as computer system 300 illustrated in FIG. 3.

[0040] As described, a database server, such as each of instances 108a-108n, is a combination of integrated software components and an allocation of computational resources (such as memory and processes) for executing the integrated software components on a processor, where the combination of the software and computational resources are used to manage a particular database, such as database 112. Among other functions of database management, a database server typically governs and facilitates access to database 112 by processing requests from clients to access the database 112. Therefore, connection pool manager 114, which manages connections to database server 104, can be implemented as a processing layer between clients 102a-102n and instances 108a-108n, which manage access to database 112. Instances 108a-108n, in conjunction with respective nodes 110a-110n, host services, described hereafter.

[0041] Services 106

[0042] As described, services are a logical abstraction for managing workloads. A service, such as service 106a-106n, is the performance of work of a particular type or category, where the work performed is for the benefit of one or more clients of one or more servers, and where a level or quality of performance is prescribed for the particular type or category of work. The work performed as part of a service includes any use or expenditure of computer resources, including, for example, CPU processing time, storing and accessing data in volatile memory and read and writes from and/or to persistent storage (i.e., IO processing), and use of network or bus bandwidth.

[0043] In general, a service is work that is performed by a database server and typically includes the work performed to process and/or compute queries that require access to a particular database. The term query as used herein refers to a statement that conforms to a database language, such as SQL, and includes statements that specify operations to add, delete, or modify data and create and modify database objects, such as tables, objects views, and executable routines.

[0044] A single database may support many services. For a non-limiting example, services may be divided into a FIN service and a PAY service, where the FIN and PAY services share the clustered database. The FIN service is the database service performed by server 104 for a FIN application. Typically, the FIN service involves accessing database objects on database 112 that store data for FIN applications. The PAY service is the service performed by server 104 for PAY applications. Typically, the PAY service involves accessing database objects on database 112 that store data for PAY applications.

US 2005/0262183 A1

Nov. 24, 2005

4

[0045] Database instances of a database cluster are allocated to support one or more services. When a database instance (or node) is allocated to perform a service, the database instance is referred to herein as hosting, running, or providing the service, and the service is referred to herein as running or being placed on the database instance.

[0046] Services can be provided by one or more database server instances. The service on each instance is referred to as a master of that service. Thus, multiple server instances may work together to provide a service to a client. In FIG. 1, service 106a (e.g., FIN) is depicted, with dashed brackets, as being provided by instance 108a, service 106b (e.g., PAY) is depicted as being provided by instances 108a and 108b, and service 106n is depicted as being provided by instances 108a-108n.

[0047] Connection Pool Manager 114 and Connection Pools 116

[0048] A connection pool is a named group of identical connections to the database that are created when the connection pool is registered. Applications that interact with the database borrow connections from the pool, use the connections to execute work requests, and then return the connections to the pool. Connection pools operate by creating connections infrequently and keeping the connections in the connection pool for long duration. Work requests come into the connection pool with high frequency, borrow these connections, and exist for relatively short duration.

[0049] As described herein, work requests can be distributed across the instances of a cluster offering a service, using (1) the service performance, e.g., as represented by response time and/or throughput; (2) available resource capacity, e.g., resource profile; and (3) service resource needs, e.g., service demand. In one embodiment, balancing of work requests occurs at two different times—at connect time and at runtime. These are referred to as Connection Load Balancing and Runtime Connection Load Balancing.

[0050] The connection pool manager 114 is a software component, which manages connection pools 116a-116n and requests for connections for sessions with server 104, including the routing of connection requests to the appropriate connection pool 116a-116n, based on the pool setup and configuration. Each connection pool 116a-116n is a set of connections to database sessions. Each session, for which communications between a client 102a-102n and an instance 108a-108n are transmitted through one of the connections from a respective connection pool 116a-116n, is considered a session with an instance 108a-108n.

[0051] Each connection can be instantiated as a connection object and for each connection object, the following information is recorded when a database session connection is established: (1) the service that the session is using; (2) the name of the node on which the associated service is available and to which this connection belongs; (3) the unique name of the database in use; and (4) the name of the instance to which this connection's database session belongs. Thus, the location (at times referred to as the "signature") of each session is uniquely identified. The manner in which the signature is recorded may vary from implementation to implementation. For non-limiting examples, a signature may be recorded to a bulletin board mechanism that is accessible to various subscribers, or the signature may be stored in an

indexed table or a hash table. The signature is also returned to the connection in various ways. In one embodiment, a handle is used as part of a connection conversation. Detailed actions that various subscribers may perform in response to notification events are beyond the scope of this description.

[0052] Generally, connection pool manager 114 is responsible for creation, maintenance and removal of connections and connection pools 116a-116n. For example, connection pool manager 114 may be used to manage connection pools 116a-116n through use of a set of APIs. In addition, connection pool manager 114 binds a connection pool 116a-116n to its corresponding data source object. In this context, a data source object is an abstraction of an interface to the database 112. Hence, binding a connection pool to a data source object may include mapping the connection pool to the data source object via a URL that identifies the associated service and the port through which the database 112 is accessed for the service.

[0053] A particular service, such as any of services 106a-106n, may be provided by one or more instances 108a-108n. A particular service is associated with a particular connection pool 116a-116n that consists of multiple connections to the one or more instances. For example, service 106a may be provided through connections from connection pool 106a; service 106b may be provided by connections from connection pool 106b; and service 106n may be provided by connections from connection pool 106n. Such an association, or mapping, can be implemented by setting and maintaining URLs that identify the instances 108a-108n that can provide a given service 106a-106n, where such an association is maintained by the connection pool manager 114. The database publishes many services. The connection pools 116a-116n can use any service that is identified, such as by the URL or a connect string.

[0054] Notification Service

[0055] In general, a daemon is a process that runs in the background and that performs a specified operation at pre-defined times or in response to certain events. In general, an event is an action or occurrence whose posting is detected by a process. Notification service daemon 118 is a process that receives alert and advisory information from server 104, such as from background manageability monitors that handle automatic management functions of instances 108a-108n. The server 104 posts service level performance events automatically and periodically, for subscribers to such events, such as runtime connection load balancing clients 102a-102n. In one embodiment, service level performance events are posted periodically based on the service request rate.

[0056] Notification service daemon 118 has a publisher-subscriber relationship with event handler 120 through which service performance information that is received by daemon 118 from server 104 is transmitted as work distribution advisory events to event handler 120. In general, an event handler is a function or method containing program statements that are executed in response to an event. In response to receiving event information from daemon 118, event handler 120 at least passes along the event type and attributes, which are described herein. A single event handler 120 is depicted in FIG. 1 as serving all subscribers. However, different event handlers may be associated with different subscribers. The manner in which handling of advi-

US 2005/0262183 A1

Nov. 24, 2005

5

sory events is implemented by various subscribers to such events is unimportant, and may vary from implementation to implementation.

[0057] For a non-limiting example, notification service daemon 118 may use the Oracle Notification System (ONS) API, which is a messaging mechanism that allows application components based on the Java 2 Platform, Enterprise Edition (J2EE) to create, send, receive, and read messages.

[0058] Load Balancing in Clustered Computing Systems

[0059] In an embodiment, the connection pool manager performs (1) connection load balancing (e.g., distribution of connections) and (2) runtime connection load balancing (e.g., distribution of work), based on service level performance metrics associated with server instances that provide a service that performs work on a node of a clustered computing system. For example, with reference to FIG. 1, work requests coming from clients 102a-102n may be distributed by connection pool manager 114, via connections, among nodes 10a-10n that host instances 108a-108n that offer services 106a-106n that perform requested work. The distribution of the requested work is based on service level performance metrics that are associated with the active instances that are providing the particular service(s) that perform the requested work.

[0060] Service Measures

[0061] A performance metric is data that indicates the quality of performance realized by services, for one or more resources. A performance metric of a particular type that can be used to gauge a characteristic or condition that indicates a service level of performance is referred to herein as a service measure. Service measures include, for example, completed work per second, elapsed time for completed calls, resource consumption and resource demand, wait events, and the like. Service measures are automatically maintained, for every service.

[0062] Approaches to generating performance metrics, including service-based performance metrics on which service measures are based, which may be used for load balancing across a database cluster, are described in U.S. patent application Ser. No. 10/917,715 filed on Aug. 12, 2004, entitled "Managing Workload By Service" and U.S. patent application Ser. No. _____ (Docket No. 50277-2336).

[0063] For example, a background process may generate performance metrics from performance statistics that are generated for each session and service hosted on a database instance. Like performance metrics, performance statistics can indicate a quality of performance. However, performance statistics, in general, include more detailed information about specific uses of specific resources. Performance statistics include, for example, how much time CPU time was used by a session, the throughput of a call, the number of calls a session made, the response time required to complete the calls for a session, how much CPU processing time was used to parse queries for the session, how much CPU processing time was used to execute queries, how many logical and physical reads were performed for the session, and wait times for input and output operations to various resources, such as wait times to read or write to a particular set of data blocks. Performance statistics gener-

ated for a session are aggregated by services and service subcategories (e.g. module, action) associated with the session.

[0064] Connection Load Balancing-Generally

[0065] A good implementation of an application that connects to a clustered computing system connects once to the system and stays connected. Since connections are relatively static, a process for balancing connections across a service that is offered by multiple server instances should not depend on metrics that vary widely during the lifetime of the connection.

[0066] Service Performance Metrics for use in Connection Load Balancing

[0067] Therefore, in one embodiment, metrics regarding instances providing a particular service are available for use when distributing the assignment, or allocation, of connections from a given connection pool among instances providing the particular service: (1) session count, per instance; (2) run queue length of the node hosting the instance; (3) service level quality weighted by available capacity; and (4) weighted session count, per service, which are described in U.S. patent application Ser. No. 10/917,715. Once connections are allocated to a particular instance, the connections are ready to establish sessions with server instances in the computing system and to accept work requests for servicing. Further, in one embodiment, such work requests are routed to server instances via the connections, based on runtime load balancing performance advisories (e.g., work distribution percentages), such as those described in U.S. patent application Ser. No. _____ (Docket No. 50277-2336).

[0068] For services that are uniformly distributed across a clustered computing system, and for nodes having similar processing capacities, the session count per instance is useful in evenly distributing connections across the system. For services that use a subset of the instances of a clustered computing system, and for nodes having different processing capacities, service quality weighted by available capacity is useful for distributing connections across the system by placing more potential sessions on the nodes that have less current processing load, at the time of connection creation.

[0069] For all services, including those having different priorities and using all or a subset of instances, and for nodes having any processing capacity, the goodness of the service is a ranking of the relative quality of service that the service is experiencing at an instance. In this embodiment, a performance grade associated with a particular service is, generally, a normalized ranking that compares the service response time and/or throughput of a service on an instance. Furthermore, the performance grades may also consider instance states, such as a state in which access to an instance is restricted. Examples of goodness rankings are "excellent," "average," "violating," and "restricted."

[0070] For all services, including those having different priorities and using all or a subset of instances, and for nodes having any processing capacity, the weighted session count by service of the service is a ranking of the power of the node to serve that service. The weighting compares the session count for the service to the node power. It also considers states such as a state in which access to an instance is restricted.

US 2005/0262183 A1

Nov. 24, 2005

6

[0071] Ideally, connections are created infrequently and stay in a connection pool for a relatively long duration. Therefore, the run queue length and the goodness value of a service when the connection is created at, for example, 9:00 A.M., has little to no relevance to how good this connection is at 5:00 P.M. The run queue length and goodness metrics are more beneficial when computed frequently, for example, as close to use-time as reasonably possible, and/or when used in combination with other methods that gravitate the work requests to the best instances.

[0072] A session lifetime by service metric allows a determination of whether to consider static metrics or dynamic metrics for connection load balancing. The weighted sessions by service metric is useful in creating connections across a clustered computing system, that are long-lived. This is typical for connection pools and SQL*Forms sessions. The weighted sessions by service metric is insensitive to time and minimizes the need to relocate sessions as the profile of the system changes.

[0073] Runtime Connection Load Balancing

[0074] Runtime connection load balancing techniques are used when selecting connections from a connection pool to execute work requests. For connection pools that allocate connections to services offered by one instance only, the first available connection in the pool is adequate for use in routing and servicing a work request. However, when connection pools allocate connections to services that span multiple instances, using a policy such as service metrics distributes the work requests across instances that are serving the service well, and avoids sending work to slow, hung, failed and restricted instances.

[0075] For example, the service time, service throughput, and resource consumption are available for each service at each instance. Just as this data may be used by the connection pool manager to allocate connections across a service, the connection pool manager may use these data when selecting a connection from the pool to best serve a work request.

[0076] Connection load balancing is used only when connections are created. Since connections should be created infrequently, connection load balancing benefits from use of metrics that do not vary over time. Conversely, runtime connection load balancing is used whenever a work request (e.g., request for execution of a database statement) is made to a connection pool for a connection. This is a very frequent activity. Hence, the metric(s) used for runtime connection load balancing should be sensitive to the current performance of services across the system. The shorter the duration of the work request, the more sensitive the metric should be. Therefore, the elapsed time for completed calls (related to service time) and completed work (related to service throughput), in conjunction with the available capacity of a node (i.e., current node profile) and the resources required by a service (i.e., demand), are very good metrics for runtime connection load balancing purposes.

[0077] Service Events

[0078] In one embodiment, each connection pool manager 116a-116n (FIG. 1) subscribes to performance grade advisory events, where the event payload contains a service performance grade for each instance offering the service. Each event is relative to a particular service. In an embodi-

ment, the posting process acquires these data once for all active services, and then posts an event per service.

[0079] As discussed, notification service daemon 118 has a publisher/subscriber relationship with event handler 120, through which certain event information that is received by daemon 118 from database server 104 is transmitted to event handler 120. In response to receiving event information from daemon 118, event handler 120 invokes a method of connection pool manager 114, passing along the event type and property, which are described hereafter.

[0080] In one embodiment, a message format that is used for communicating service performance information in an event payload, comprises name/value pairs. Specifically, a service event may comprise the following:

[0081] Event Type=performance grades (e.g., "Database/Service/Grades/\$service_name");

[0082] Version=version number of the publication protocol;

[0083] Database name=unique name identifying database;

[0084] Grades tuple (repeating for plurality of instances offering service)=instance name, flag (i.e., GOOD, VIOLATING, NODATA, UNKNOWN), grade; and

[0085] Timestamp=time event was calculated at the server.

[0086] Runtime Connection Load Balancing Based on Performance Grade

[0087] In an embodiment, runtime balancing of system load (e.g., work being performed by the system) is based on service performance grade advisories, e.g., using the grades tuple (instance name, flag, grade) as the basis for making decisions. As discussed, these data are published to remote listeners that subscribe to the data publication. By providing this data to a connection pool manager client, better overall service is obtained by proactively routing work to the instances that best serve the work, rather than relocating sessions later in a reactive manner.

[0088] According to one embodiment, a method for runtime connection load balancing using service performance comprises (1) connection load balancing, i.e., allocating connections in a connection pool (which is associated with a particular service) to relatively well-performing instances ("gravitating") based, for example, on service goodness value and possibly a service delta value; and (2) runtime connection load balancing, i.e., in response to receiving work requests, routing the work requests to the currently well-performing instances based, for example, on service goodness percentages.

[0089] Server Functionality

[0090] In one embodiment, every few seconds every instance updates the moving average of its local service metrics and posts this to a master instance. The aggregated data and percentages are calculated internally at the master, and then posted to the event system, as described in U.S. patent application Ser. No. _____ (Docket No. 50277-2336).

US 2005/0262183 A1

Nov. 24, 2005

7

[0091] Connection Pool Functionality**[0092]** I. Connection Retrieval

[0093] The server posts performance grade tuples to the connection pool periodically, where the tuples provided are for each database service. The grade tuples include grades, which represent percentage of work to send to that instance, as well as a flag indicating the state of the instance. The sum of the percentages for each service is 100%.

[0094] The Connection Pool retrieves connections to instances based on the four possible flag values: GOOD, UNKNOWN, VIOLATING, NO_DATA. The first three flag values indicate that the instance percentages are valid and should be honored. The Connection Pool will retrieve connections to these instances. The NO_DATA flag indicates that the instance is not responding, in which case the Connection Pool will not retrieve any connections to an invalid instance.

[0095] In one embodiment, the Connection Pool satisfies a connection request to a given service by performing the following steps:

- [0096]** (1) Compute the sum of the percentages for the valid instances providing the service.
- [0097]** (2) Compute a pseudo random number greater than or equal to zero and less than the sum computed in step 1.
- [0098]** (3) Arrange the instances in some arbitrary order. Set the value TOTAL to zero. Consider the first instance.
- [0099]** (4) Add the percentage for the instance to TOTAL.
- [0100]** (5) If the pseudo random number is less than TOTAL, select the instance. Otherwise, consider the next instance and go to step 4.
- [0101]** (6) Retrieve a connection from the selected instance from the connection pool.

[0102] II. Connection Gravitation

[0103] A purpose of a connection pool, such as connection pools 116a-116n, is to have connections that are ready to use when connection requests are received. Consistently and periodically gravitating the connections from a given connection pool toward the well-performing ("healthy") instances helps to ensure that all connections are good to use when work requests are received. A connection pool manager, such as connection pool manager 114, typically releases connections that are idle in the connection pool for a specified expiration period, and some connection pool managers release connections in the connection pool because the connections have been in the pool for some time.

[0104] Generally, the process of gravitating connections toward the healthy instances may comprise any of (1) determining which connections to retire, (2) determining which connections to keep and, (3) when connections are added, ensuring that these new connections are allocated to healthy instances. Previously, the connections were retired randomly and at arbitrary times, i.e., there was no knowledge as to which connections were good connections rela-

tive to their ability to facilitate performance of work, and no knowledge of when retiring connections was needed.

[0105] Which instance a connection is connected to is determined by the metrics current at the moment when the connection is created. As a result, the number of connections to each instance approximates the percentages in the metrics. Over time the metrics change, but the connections to the instance to which it is connected typically would not change. As a result, the number of connections to each instance would no longer approximate the percentages. This results in too many connections with low percentages and too few connections to instances with high percentages. Hence, connection gravitation gradually rectifies this problem by selectively retiring connections to an instance.

[0106] Connections are selected for retirement based on the following two ratios:

$$\text{Attempted Connection Ratio} = (\text{Request count per instance}) / (\text{Total request count});$$

$$\text{Connection Ratio} = (\text{Connection count per instance}) / (\text{Total connections in the pool}).$$

[0107] Retirement criteria for an instance is met when there are far more connections available to an instance than the metrics percentage indicates for that instance. If for a given instance the Connection Ratio is much greater than the Attempted Connection Ratio, then instances to that connection can be retired. In one embodiment, once an instance is chosen for retiring connections, about 25% of available connections are actually retired. This allows for a gradual gravitation over time.

[0108] Method for Routing Work Requests to a Clustered Computing System

[0109] FIG. 2 is a flow diagram that illustrates a method for routing work requests to a clustered computing system, according to an embodiment of the invention. The method may be performed by any sequence(s) of one or more instructions (e.g., a software module) which, when executed, cause one or more processors to perform steps illustrated in the blocks of FIG. 2. For example, the method may be performed by connection pool manager 114 in the context of a database server 104. However, the techniques embodied in the method of FIG. 2 may be performed by any implementation that manages and/or distributes work across a clustered computing system by providing connections to a component or module that processes work requests, for routing of such requests.

[0110] At block 202, a request is received for a connection to the clustered computing system for performance of work. For example, connection pool manager 114 receives a request from a client 102a, for a connection to database server 104. The request may be in the form of an explicit request for a connection, a request for work, or any other type of request that would require a session with, and therefore a connection to, the clustered computing system.

[0111] At block 204, a pool of connections is identified that is associated with a service that performs the work for which the connection is requested. For example, connection pool manager 114 determines that service 106b performs the requested work and identifies that connection pool 116b serves connections to the service 106b. The service that performs the work may be identified in a number of ways, such as based on the nature of the request or based on

US 2005/0262183 A1

Nov. 24, 2005

8

information contained in the request that specifies the service. A mapping between services and associated connection pools may be accessed to identify the specific pool that is associated with the identified service.

[0112] At block 206, a connection to an instance that provides the requested service is selected from the identified connection pool, based on metrics related to the operational performance of the service on the identified instances. The selected connection is used for routing a request to the particular node/instance/service for performance of the work. In one embodiment, the connection is selected based on the service grade, described above. For example, the selection is based on the substantially current service response time or throughput for each of the instances providing the service, characterized by the grades tuple (i.e., instance name, flag, grade (i.e., work distribution percentage)), or based on a subset thereof.

[0113] Implementation Mechanisms

[0114] The approach for runtime connection load balancing of work across a clustered computing system, as described herein, may be implemented in a variety of ways and the invention is not limited to any particular implementation. The approach may be integrated into a system or a device, or may be implemented as a stand-alone mechanism. Furthermore, the approach may be implemented in computer software, hardware, or a combination thereof.

[0115] Hardware Overview

[0116] FIG. 3 is a block diagram that depicts a computer system 300 upon which an embodiment of the invention may be implemented. Computer system 300 includes a bus 302 or other communication mechanism for communicating information, and a processor 304 coupled with bus 302 for processing information. Computer system 300 also includes a main memory 306, such as a random access memory (RAM) or other dynamic storage device, coupled to bus 302 for storing information and instructions to be executed by processor 304. Main memory 306 also may be used for storing temporary variables or other intermediate information during execution of instructions to be executed by processor 304. Computer system 300 further includes a read only memory (ROM) 308 or other static storage device coupled to bus 302 for storing static information and instructions for processor 304. A storage device 310, such as a magnetic disk or optical disk, is provided and coupled to bus 302 for storing information and instructions.

[0117] Computer system 300 may be coupled via bus 302 to a display 312, such as a cathode ray tube (CRT), for displaying information to a computer user. An input device 314, including alphanumeric and other keys, is coupled to bus 302 for communicating information and command selections to processor 304. Another type of user input device is cursor control 316, such as a mouse, a trackball, or cursor direction keys for communicating direction information and command selections to processor 304 and for controlling cursor movement on display 312. This input device typically has two degrees of freedom in two axes, a first axis (e.g., x) and a second axis (e.g., y), that allows the device to specify positions in a plane.

[0118] The invention is related to the use of computer system 300 for implementing the techniques described herein. According to one embodiment of the invention, those

techniques are performed by computer system 300 in response to processor 304 executing one or more sequences of one or more instructions contained in main memory 306. Such instructions may be read into main memory 306 from another machine-readable medium, such as storage device 310. Execution of the sequences of instructions contained in main memory 306 causes processor 304 to perform the process steps described herein. In alternative embodiments, hard-wired circuitry may be used in place of or in combination with software instructions to implement the invention. Thus, embodiments of the invention are not limited to any specific combination of hardware circuitry and software.

[0119] The term "machine-readable medium" as used herein refers to any medium that participates in providing instructions to processor 304 for execution. Such a medium may take many forms, including but not limited to, non-volatile media, volatile media, and transmission media. Non-volatile media includes, for example, optical or magnetic disks, such as storage device 310. Volatile media includes dynamic memory, such as main memory 306. Transmission media includes coaxial cables, copper wire and fiber optics, including the wires that comprise bus 302. Transmission media can also take the form of acoustic or light waves, such as those generated during radio-wave and infra-red data communications.

[0120] Common forms of machine-readable media include, for example, a floppy disk, a flexible disk, hard disk, magnetic tape, or any other magnetic medium, a CD-ROM, any other optical medium, punchcards, papertape, any other physical medium with patterns of holes, a RAM, a PROM, and EPROM, a FLASH-EPROM, any other memory chip or cartridge, a carrier wave as described hereinafter, or any other medium from which a computer can read.

[0121] Various forms of computer readable media may be involved in carrying one or more sequences of one or more instructions to processor 304 for execution. For example, the instructions may initially be carried on a magnetic disk of a remote computer. The remote computer can load the instructions into its dynamic memory and send the instructions over a telephone line using a modem. A modem local to computer system 300 can receive the data on the telephone line and use an infra-red transmitter to convert the data to an infra-red signal. An infra-red detector can receive the data carried in the infra-red signal and appropriate circuitry can place the data on bus 302. Bus 302 carries the data to main memory 306, from which processor 304 retrieves and executes the instructions. The instructions received by main memory 306 may optionally be stored on storage device 310 either before or after execution by processor 304.

[0122] Computer system 300 also includes a communication interface 318 coupled to bus 302. Communication interface 318 provides a two-way data communication coupling to a network link 320 that is connected to a local network 322. For example, communication interface 318 may be an integrated services digital network (ISDN) card or a modem to provide a data communication connection to a corresponding type of telephone line. As another example, communication interface 318 may be a local area network (LAN) card to provide a data communication connection to a compatible LAN. Wireless links may also be implemented. In any such implementation, communication interface 318

US 2005/0262183 A1

Nov. 24, 2005

9

sends and receives electrical, electromagnetic or optical signals that carry digital data streams representing various types of information.

[0123] Network link 320 typically provides data communication through one or more networks to other data devices. For example, network link 320 may provide a connection through local network 322 to a host computer 324 or to data equipment operated by an Internet Service Provider (ISP) 326. ISP 326 in turn provides data communication services through the world wide packet data communication network now commonly referred to as the "Internet" 328. Local network 322 and Internet 328 both use electrical, electromagnetic or optical signals that carry digital data streams. The signals through the various networks and the signals on network link 320 and through communication interface 318, which carry the digital data to and from computer system 300, are exemplary forms of carrier waves transporting the information.

[0124] Computer system 300 can send messages and receive data, including program code, through the network(s), network link 320 and communication interface 318. In the Internet example, a server 330 might transmit a requested code for an application program through Internet 328, ISP 326, local network 322 and communication interface 318.

[0125] The received code may be executed by processor 304 as it is received, and/or stored in storage device 310, or other non-volatile storage for later execution. In this manner, computer system 300 may obtain application code in the form of a carrier wave.

[0126] Extensions and Alternatives

[0127] Alternative embodiments of the invention are described throughout the foregoing description, and in locations that best facilitate understanding the context of the embodiments. Furthermore, the invention has been described with reference to specific embodiments thereof. It will, however, be evident that various modifications and changes may be made thereto without departing from the broader spirit and scope of the invention. For example, embodiments of the invention are described herein in the context of a database server; however, the described techniques are applicable to any distributed computing system over which system connections are allocated or assigned, such as with a system configured as a computing cluster or a computing grid. Therefore, the specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense.

[0128] In addition, in this description certain process steps are set forth in a particular order, and alphabetic and alphanumeric labels may be used to identify certain steps. Unless specifically stated in the description, embodiments of the invention are not necessarily limited to any particular order of carrying out such steps. In particular, the labels are used merely for convenient identification of steps, and are not intended to specify or require a particular order of carrying out such steps.

What is claimed is:

1. A computer-implemented method for routing work requests to a computing system that comprises a plurality of

computing nodes that each hosts a server instance that provides a service that performs work, the method comprising:

receiving a request for a connection to said computing system for requesting said service;

identifying a pool of connections, to said computing system, that is associated with said service, wherein each connection in said pool is associated with a particular server instance; and

selecting, from said pool, a connection that is associated with said service, to route a request for performance of work associated with said service;

wherein said selecting is based on substantially current performance information for said instances that offer said service that is associated with said selected connection.

2. The method of claim 1, wherein the performance information for each of said instances that offer said service includes a value representing a percentage of work to route to said instance, and wherein selecting a connection to route the request comprises:

(a) computing the sum of said percentages for valid instances providing said service;

(b) computing a pseudo-random number that is greater than or equal to zero and less than said sum of said percentages;

(c) set a variable value to zero;

(d) randomly order said valid instances;

(e) select the first instance of said ordered valid instances;

(f) sum said percentage for said first instance and said variable value; and

(g) if said pseudo-random number is less than said variable value, then select the first instance, otherwise select the next instance of said ordered valid instances and perform steps (f) and (g) until a valid instance is selected.

3. The method of claim 2, further comprising

retrieving, from said pool, said selected connection to route the request.

4. The method of claim 1, further comprising:

selecting a connection, from said pool, to retire from said pool.

5. The method of claim 4, wherein selecting a connection to retire includes:

selecting an instance from which to retire said connection by computing, for each instance that offers said service,

a first ratio of the number of requests for a connection to said instance over a total number of requests for said service;

a second ratio of the number of connections to said instance over a total number of connections in said pool; and

selecting said instance from which to retire said connection based on the difference between said second ratio and said first ratio.

US 2005/0262183 A1

Nov. 24, 2005

10

6. The method of claim 1,

wherein said performance information comprises a characterization of any one or more of (a) a response time of said service and (b) a throughput of said service, associated with a respective instance providing said service on a node of said computing system.

7. A machine-readable medium carrying one or more sequences of instructions which, when executed by one or more processors, causes the one or more processors to perform the method recited in claim 1.

8. A machine-readable medium carrying one or more sequences of instructions which, when executed by one or more processors, causes the one or more processors to perform the method recited in claim 2.

9. A machine-readable medium carrying one or more sequences of instructions which, when executed by one or more processors, causes the one or more processors to perform the method recited in claim 3.

10. A machine-readable medium carrying one or more sequences of instructions which, when executed by one or more processors, causes the one or more processors to perform the method recited in claim 4.

11. A machine-readable medium carrying one or more sequences of instructions which, when executed by one or more processors, causes the one or more processors to perform the method recited in claim 5.

12. A machine-readable medium carrying one or more sequences of instructions which, when executed by one or more processors, causes the one or more processors to perform the method recited in claim 6.

13. A computer-implemented method for managing connections to a computing system that comprises a plurality of

computing nodes that each hosts a server instance that provides a service that performs work, the method comprising:

periodically gravitating connections to server instances, from a plurality of server instances executing on said plurality of computing nodes, that are performing well relative to other server instances from said plurality of server instances; and

wherein said gravitating is based on service performance information for instances that offer said service that is associated with connections being gravitated.

14. The method of claim 13, further comprising:

in response to a request for a connection to said computing system for requesting said service, selecting a connection that is associated with said service, to route a request for performance of work associated with said service;

wherein said selecting is based on substantially current performance information for said instances that offer said service that is associated with said selected connection.

15. A machine-readable medium carrying one or more sequences of instructions which, when executed by one or more processors, causes the one or more processors to perform the method recited in claim 13.

16. A machine-readable medium carrying one or more sequences of instructions which, when executed by one or more processors, causes the one or more processors to perform the method recited in claim 14.

* * * * *

Exhibit 33

**THIS EXHIBIT HAS BEEN
REDACTED IN ITS ENTIRETY**

Exhibit 34

**THIS EXHIBIT HAS BEEN
REDACTED IN ITS ENTIRETY**

Exhibit 35

**THIS EXHIBIT HAS BEEN
REDACTED IN ITS ENTIRETY**

Exhibit 36

**THIS EXHIBIT HAS BEEN
REDACTED IN ITS ENTIRETY**

Exhibit 37

**THIS EXHIBIT HAS BEEN
REDACTED IN ITS ENTIRETY**

Exhibit 38

05-25-01

GP/2182/H

ATTORNEY DOCKET NO.
066241.0119

PATENT APPLICATION
09/234,048

1



IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

#9/B
RECEIVED
MAY 30 2001
Technology Center 2100

In re Application of: Keith Lowery, et al.
Serial No.: 09/234,048
Filing Date: January 19, 1999
Group Art Unit: 2182
Examiner: Rehana Perveen
Title: SYSTEM AND METHOD FOR MANAGING
DYNAMIC WEB PAGE GENERATION REQUESTS

BOX: Patent Application

Honorable Assistant Commissioner
of Patents
Washington, D.C. 20231

Dear Sir:

RESPONSE TO OFFICE ACTION

In response to the Office Action mailed February 23, 2001, Applicants respectfully request the Examiner to reconsider the rejection of the claims in view of the following comments as set forth below. The Claims have not been amended.

IN THE SPECIFICATION

In the Title, please delete "SYSTEM AND METHOD FOR MANAGING DYNAMIC WEB PAGE GENERATION REQUESTS" and replace with -- SYSTEM AND METHOD FOR RESPONDING TO REQUESTS ASSOCIATED WITH DYNAMIC WEB PAGE GENERATION --.

DAL01:604935.1

EPIC000489

ATTORNEY DOCKET NO.
066241.0119

PATENT APPLICATION
09/234,048

2

IN THE CLAIMS

For the convenience of the Examiner, all pending claims of the Application are reproduced below.

Claims 1-16 have previously been cancelled.

17. A computer-implemented method for managing a dynamic Web page generation request to a Web server, said computer-implemented method comprising the steps of:

routing a request from a Web server to a page server, said page server receiving said request and releasing said Web server to process other requests wherein said routing step further includes the steps of:

intercepting said request at said Web server and routing said request to said page server;

processing said request, said processing being performed by said page server while said Web server concurrently processes said other requests; and

dynamically generating a Web page in response to said request, said Web page including data dynamically retrieved from one or more data sources.

18. The computer-implemented method in Claim 17 wherein said step of routing said request includes the steps of:

routing said request from said Web server to a dispatcher; and

dispatching said request to said page server.

19. The computer-implemented method in Claim 17 wherein said step of processing said request includes the step of identifying said one or more data sources from which to retrieve said data.

DAL01:604935.1

EPIC000490

ATTORNEY DOCKET NO.
066241.0119

PATENT APPLICATION
09/234,048

3

20. The computer-implemented method in Claim 17 wherein said step of dynamically generating said Web page includes the step of dynamically retrieving said data from said one or more data sources.

21. The computer-implemented method in Claim 17 wherein said step of processing said request includes the step of said page server maintaining a connection cache to said one or more data sources.

22. The computer-implemented method in Claim 17 wherein said step of processing said request includes the step of logging into said one or more data sources.

23. The computer-implemented method in Claim 17 wherein said step of dynamically generating said Web page includes the step of maintaining a page cache containing said Web page.

24. The computer-implemented method in Claim 17 wherein said page server includes tag-based text templates for configuring said Web page.

25. The computer-implemented method in Claim 24 wherein said step of processing said request further includes the step of inserting said dynamically retrieved data from said one or more data sources into said tag-based text templates.

26. The computer-implemented method in Claim 24 wherein at least one of said tag-based text templates drives a format of the data dynamically retrieved from said one or more data sources in response to said request.

DAL01:604935.1

EPIC000491

ATTORNEY DOCKET NO
066241.0119

PATENT APPLICATION
09/234,048

4

27. The computer-implemented method in Claim 24 wherein said tag-based text templates include HTML templates.

28. The computer-implemented method in Claim 17 wherein said step of processing said request further includes the step of dynamically updating data at said one or more data sources.

29. The computer-implemented method in Claim 17 wherein said step of processing said request further includes the step of processing an object handling extension.

30. The computer-implemented method in Claim 29 wherein said object handling extension is an OLE extension.

DAL01:604935.1

EPIC000492

ATTORNEY DOCKET NO.
066241.0119

PATENT APPLICATION
09/234,048

5

31. A computer-implemented method comprising the steps of:

transferring a request from an HTTP-compliant device to a page server, said page server receiving said request and releasing said HTTP-compliant device to process other requests wherein said transferring step further includes the steps of:

intercepting said request at said HTTP-compliant device and transferring said request to said page server;

processing said request, said processing being performed by said page server while said HTTP-compliant device concurrently processes said other requests; and

dynamically generating a page in response to said request, said page including data dynamically retrieved from one or more data sources.

32. The computer-implemented method in Claim 31 wherein said step of transferring said request includes the steps of:

transferring said request from said HTTP-compliant device to a dispatcher; and

dispatching said request to said page server.

33. The computer-implemented method in Claim 31 wherein said step of processing said request includes the step of identifying said one or more data sources from which to retrieve said data.

34. The computer-implemented method in Claim 31 wherein said step of dynamically generating said page includes the step of dynamically retrieving said data from said one or more data sources.

DAL01:604935.1

EPIC000493

ATTORNEY DOCKET NO.
066241.0119

PATENT APPLICATION
09/234,048

6

35. The computer-implemented method in Claim 31 wherein said step of processing said request includes the step of said page server maintaining a connection cache to said one or more data sources.

36. The computer-implemented method in Claim 31 wherein said step of processing said request includes the step of logging into said one or more data sources.

37. The computer-implemented method in Claim 31 wherein said step of dynamically generating said page includes the step of maintaining a page cache containing said page.

38. The computer-implemented method in Claim 31 wherein said page server includes tag-based text templates for configuring said page.

39. The computer-implemented method in Claim 38 wherein said step of processing said request further includes the step of inserting said dynamically retrieved data from said one or more data sources into said tag-based text templates.

40. The computer-implemented method in Claim 38 wherein at least one of said tag-based text templates drives a format of the data dynamically retrieved from said one or more data sources in response to said request.

41. The computer-implemented method in Claim 38 wherein said tag-based text templates include HTML templates.

DAL01:604935.1

EPIC000494

ATTORNEY DOCKET NO.
066241.0119

PATENT APPLICATION
09/234,048

7

42. The computer-implemented method in Claim 31 wherein said step of processing said request further includes the step of dynamically updating data at said one or more data sources.

43. The computer-implemented method in Claim 31 wherein said step of processing said request further includes the step of processing an object handling extension.

44. The computer-implemented method in Claim 43 wherein said object handling extension is an OLE extension.

DAI01:604935.1

EPIC000495

ATTORNEY DOCKET NO.
066241.0119

PATENT APPLICATION
09/234,048

8

45. A computer-implemented method comprising the steps of:
transferring a request from an HTTP-compliant device to a dispatcher;
maintaining dynamic information regarding data sources a given page server may
access;
dispatching said request to an appropriate page server based on said request and based
on said dynamic information, said page server receiving said request and releasing said HTTP-
compliant device to process other requests;
processing said request, said processing being performed by said page server while said
HTTP-compliant device concurrently processes said other requests; and
dynamically generating a page in response to said request, said page including data
dynamically retrieved from one or more data sources.

DAL01:604935.1

EPIC000496

ATTORNEY DOCKET NO.
066241.0119

PATENT APPLICATION
09/234,048

9

REMARKS

This Application has been carefully reviewed in light of the Office Action mailed February 23, 2001. At the time of the Office Action, Claims 17-45 were pending in this patent application. The Examiner rejected Claims 17-45. Thus, Claims 17-45 are now pending in the Application. Applicants respectfully request reconsideration and favorable action in this case.

IN THE TITLE

The Examiner has rejected the title of the invention as being non-descriptive of the invention to which the claims are directed. Applicants respectfully traverse this rejection, but in the interest of advancing prosecution have amended the title to read "SYSTEM AND METHOD FOR RESPONDING TO REQUESTS ASSOCIATED WITH DYNAMIC WEB PAGE GENERATION". Applicants respectfully submit that the title as amended is descriptive of the invention to which the claims are directed.

IN THE DRAWINGS

Applicants respectfully request that submission of formal drawings be deferred until prior to issuance.

DOUBLE PATENTING

The Examiner has rejected claims 17-45 under the judicially created doctrine of obviousness-type double patenting as being unpatentable over Claims 1-11 of Lowery et al., U. S. Pat. No. 5,894,554, ("Lowery"). Applicants have included a Terminal Disclaimer with respect to the double patenting rejection. Applicants respectfully request withdrawal of this rejection.

SECTION 102 REJECTION

Claims 17-45 stand rejected under U.S.C. § 102(e) as being anticipated by Leaf, U.S. Pat. No. 5,754,772 ("Leaf"). Applicants respectfully traverse this rejection for at least the following reasons.

Claim 17 recites, in part, "intercepting said request at said Web server and routing said request to said page server". Leaf does not teach or suggest "intercepting said request".

DAL01:604935.1

EPIC000497

ATTORNEY DOCKET NO.
066241.0119

PATENT APPLICATION
09/234,048

10

Instead, Leaf teaches that the web server routes the request directly to the transaction gateway client. Leaf, col. 4, lines 55-57. Leaf does not teach or suggest "intercepting said request at said Web server" because merely routing a request from a web server to the transaction gateway does not involve interception.

Therefore, for at least this reason, Claim 17 is patentable over Leaf. Thus, Applicants respectfully request allowance of Claim 17.

Dependent Claims 18-30 depend from independent Claim 17, shown above to be allowable. Claims 18-30 are allowable as depending from an allowable base claim and as defining further distinctions over the cited reference.

In particular, Claim 21 recites, in part, "maintaining a connection cache to said one or more data sources" and Claim 23 recites, in part, "maintaining a page cache containing said Web page". The Examiner states that the elements of Claims 21 and 23 are inherent in Leaf. Office Action, p. 4. Leaf does not involve either a connection cache or a page cache. Applicants respectfully submit that neither a connection cache nor a page cache are inherent in Leaf. Applicants respectfully request that the Examiner indicate some teaching or suggestion of Leaf with respect to a page cache and a connection cache. Therefore, Claims 21 and 23 are patentable over Leaf. Thus, Applicants respectfully request allowance of Claims 21 and 23.

Also, Claim 24 recites "wherein said page server includes tag-based text templates for configuring said Web page." Applicants respectfully submit that Leaf does not teach or suggest this element of Claim 24. In contrast to the Examiner's assertion, Leaf merely teaches that the Transaction Gateway Client may format the data into an HTML document. Leaf, col. 4, lines 63-67; See also Leaf, col. 4, lines 25-27. Mere formatting of data into an HTML document does not teach or suggest a "tag-based text template for configuring said Web page". Therefore, Claim 24 is patentable over Leaf. Thus, Applicants respectfully request allowance of Claim 24.

DAL01:604935.1
066241.0119

EPIC000498

ATTORNEY DOCKET NO.
066241.0119

PATENT APPLICATION
09/234,048

11

Independent Claim 31 is also allowable at least for the reasons discussed above. In particular, Claim 31 recites "intercepting said request at said HTTP-compliant device and transferring said request to said page server". Accordingly, Applicants respectfully request that Claim 31 be allowed.

Dependent Claims 32-44 depend from independent Claim 31, shown above to be allowable. Claims 32-44 are allowable as depending from an allowable base claim and as defining further distinctions over the cited reference. In particular, Claim 35 recites "wherein said step of processing said request includes the step of said page server maintaining a connection cache to said one or more data sources." Claim 37 recites "wherein said step of dynamically generating said page includes the step of maintaining a page cache containing said page." Claim 38 recites "wherein said page server includes tag-based text templates for configuring said page." Thus, Applicants respectfully request allowance of Claims 32-44.

Independent Claim 45 recites, in part, "maintaining dynamic information regarding data sources a given page server may access". Applicants respectfully submit that Leaf does not teach or suggest this element of Claim 45. Leaf provides no teaching or suggestion of "maintaining dynamic information regarding data sources". Indeed, Leaf notes that "each transaction gateway client pre-establishes a static connection with the n-line transaction processing system." Leaf, col. 2, lines 37-40. Leaf's suggestion of static connections teaches away from "dynamic" information. Therefore, for at least this reason, Claim 45 is patentable over Leaf. Thus, Applicants respectfully request allowance of Claim 45.

DAL01:604935.1

EPIC000499

ATTORNEY DOCKET NO.
066241.0119

ATENT APPLICATION
09/234,048

12

CONCLUSION

Applicants have now made an earnest attempt to place this case in condition for immediate allowance. For the foregoing reasons and for other reasons clearly apparent, Applicants respectfully request reconsideration and allowance of Claims 17-45.

Although Applicants believe that no other fees are due, the Commissioner is hereby authorized to charge any fees or credit any overpayments to Deposit Account No. 02-0384 of Baker Botts L.L.P.

If there are matters that can be discussed by telephone to further the prosecution of this application, Applicants respectfully request that the Examiner call its attorney at the number listed below.

Respectfully submitted,
BAKER BOTTS L.L.P.
Attorneys for Applicants



Matthew B. Talpis
Reg. No. 45,152

Correspondence Address:
Matthew B. Talpis, Esq.
Baker Botts L.L.P.
2001 Ross Avenue, Suite 600
Dallas, Texas 75201-2980
Phone: 214-953-6984
Fax: 214-661-4984

Date: May 23, 2001

DAL01:604935.1

EPIC000500

ATTORNEY DOCKET # :
066241.0119

TENT APPLICATION
09/234,048



IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application of: Keith Lowery, et al.
Serial No.: 09/234,048
Filing Date: January 19, 1999
Group Art Unit: 2182
Examiner: Rehana Perveen
Title: SYSTEM AND METHOD FOR MANAGING
DYNAMIC WEB PAGE GENERATION REQUESTS

RECEIVED
MAY 30 2001
Technology Center 2100

Box: Patent Application
Assistant Commissioner for
Patents
Washington, D.C. 20231

CERTIFICATE OF MAILING BY EXPRESS MAIL

I hereby certify that the attached Response to Office Action (12 pages), an Information Disclosure Statement (1 page), a PTO Form-1449 (1 page with 1 reference), a check in the amount of \$180.00 for the stipulated filing fee, a Terminal Disclaimer (2 pages with 4 attachments), a check in the amount of \$110.00 to cover the Terminal Disclaimer filing fee, a Baker Botts L.L.P. return receipt postcard (1 card), and this Certificate of Mailing (1 page) is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 C.F.R. § 1.10 on this 23rd day of May, 2001 and is addressed to the Assistant Commissioner for Patents, Washington, DC 20231.

Willie Jiles

Willie Jiles

Express Mail Receipt
No. EL759179317US

DAL01:605946.1
066241.0119

EPIC000501

Exhibit 39

Oracle WebServer 2.0

Technical Note

March 1996

Oracle WebServer 2.0

This Technical Note is one of a series of publications from Oracle Internet Server Products, a product group within the Server Technologies Division of Oracle Corporation. This document is written for web software developers and other audiences familiar with basic Web technology. Specifically, no introduction to the Internet or Web will be given here. By reading this document, you will gain a technical understanding of how Oracle WebServer 2.0 is designed, and how you can use it to write high performance applications for the web. Many acronyms and technical terms are used in this document. For your convenience, we have included a glossary in appendix A.

Overview

Oracle WebServer 2.0 lets you publish hypermedia documents and deploy media-rich applications on the World Wide Web and corporate intranets using the industry standard HTTP protocol. Unlike mere HTTP servers, Oracle WebServer 2.0 offers a fully integrated platform for application development, mass-storage, and system and site-management, all using fully integrated state-of-the-art relational database technology and established industry standards such as HTTP, SSL, and HTML. Oracle WebServer 2.0 offers an open, extensible application development platform for the Web, enabling software developers to safely and easily add high-speed, custom server extensions to take full advantage of any and all modern technologies such as Java, distributed objects and virtual reality to name a few, without having to resort to the low-level network interfaces employed by most other HTTP servers today.

An Application Platform for the Web

Oracle WebServer 2.0 provides an open platform for developing web applications. For customers with heavy investments in the Oracle7 RDBMS, we offer the integrated PL/SQL Agent, an extension to Oracle WebServer 2.0, which allows you to develop complete web applications using Oracle7 stored procedures. For customers who wish to use Java, we offer a fully integrated Java Interpreter, enabling you to develop complete web applications using server-side Java classes. The Java Interpreter includes classes for native Oracle7 access via PL/SQL and dynamic creation of HTML documents. For customers who wish to employ their own programming language of choice, Oracle WebServer 2.0 offers an easy-to-use API allowing you to build your own

server-side plugins ranging from existing legacy applications written in COBOL to modern CORBA or COM/OLE based object-oriented systems written in for instance C++ or Smalltalk. Oracle WebServer 2.0 allows developers to easily integrate their back-end applications using our open API to the Oracle Web Request Broker.

Architecture

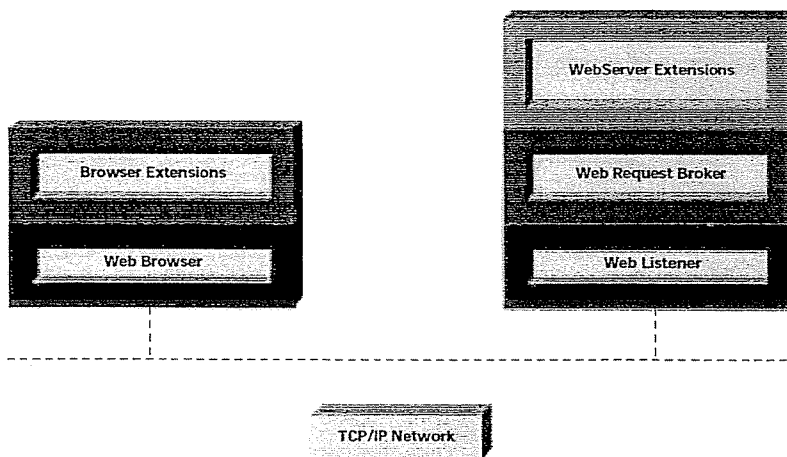
From a web browsers point of view, Oracle WebServer 2.0 looks just like any other web server. The industry standard HTTP 1.0 protocol with optional SSL based data-encryption is used to establish a short-term connection and request an object from the server. When the object is received by the client, the MIME type of the object determines how it is handled: either processed natively by the client, or handed off to a helper application or client-side plugin (extension) for further processing. This defines a complete HTTP transaction and is most often (but certainly not limited to) a request for and delivery of an HTML page, which is displayed in the browsers main viewing area.

Underneath the covers of the Oracle WebServer's HTTP server component, the Oracle Web Listener, interesting things happen to fulfill a request from a web browser. The following sections provides an in-depth description of the three major components of Oracle WebServer 2.0: the *Web Listener*, *Web Request Broker* and the *WebServer SDK*.

Oracle Web Listener

The Web Listener is responsible for delivering incoming requests to the Web Request Broker, where they are processed or dispatched to services built with the WebServer SDK.

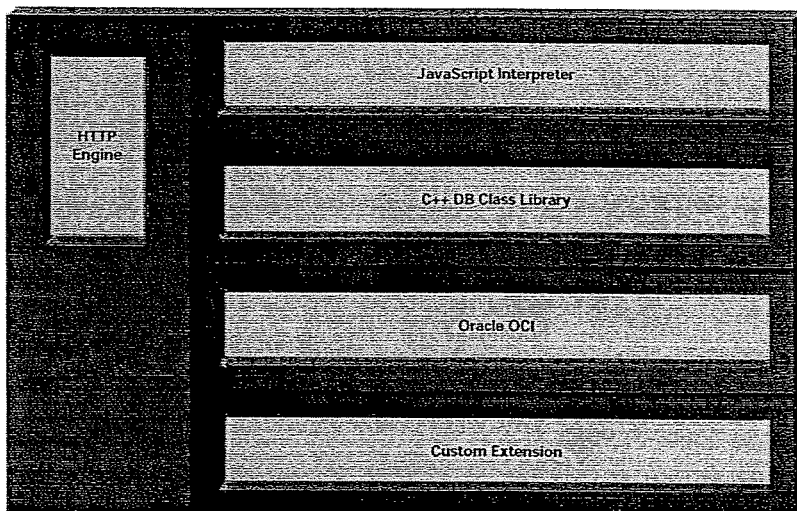
Oracle WebServer 2.0 Architecture



Oracle's HTTP Server is designed to be a very scalable, high-performance network daemon. It is implemented as a multi-threaded, single-process asynchronous engine, allowing multiple concurrent requests to be processed at the same time using standard HTTP or HTTP over SSL. The HTTP Server is the network layer component of the Oracle WebServer. It listens for incoming HTTP requests, delivers static files and runs simple CGI programs, and hands off everything else to the Web Request Broker (WRB), where a request is handled by a server extension. The first generation of websites on the Internet simply offered static HTML files to their visitors. To enable dynamic content, web designers are mainly using CGI to extend the capabilities of their web server. Although easy to use, CGI has some serious limitations, primarily the high overhead of spawning a new process for each incoming request, which causes severe degradation in scalability and throughput.

Today, several web servers have defined an API for implementing extensions to the basic HTTP engine. The typical HTTP API enables a custom extension to be linked into the HTTP server process and allows it to intercept and fulfill a client request. Examples of such APIs are Netscape NSAPI, Spyglass ADI, and Microsoft IS-API. These APIs are becoming increasingly popular because they address the serious shortcomings of the original CGI protocol.

The problem with the HTTP API approach is that all extensions are linked into the same executable and thus share the same memory, process, and signal handling space. Extensions are free to clobber each other, and a poorly designed extension can bring the entire system down. Furthermore, technical support immediately becomes a nightmare: who are you going to call when you have linked together components from 5 different vendors? What if the extension from vendor A is multi-threaded, and the extension from vendor B isn't thread-safe?



Typical HTTP server extended through API

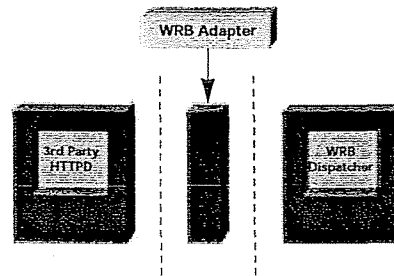
Oracle's answer to these issues is a new and improved architecture for the server, which effectively layers Oracle WebServer 2.0 into two fundamental components: a HTTP engine (the Oracle Web Listener) and a high-speed dispatch mechanism (the Web Request Broker) which routes requests to server extensions running in separate processes.

Being the industry leader in server and networking technology, Oracle fully understands how to optimize performance and throughput in an environment like the World Wide Web. Our experience with products like SQL*Net and Oracle7 show that network service daemons and information servers benefit from different internal architectures. Most important, since the network itself is the largest bottle-neck, the network listener must be lean and fast, and not carry any extra overhead to its main function.

Today, several hundred web server implementations are in use on the Internet. Ranging from a few lines of Perl code to full-blown commercial products on just about every imaginable operating system, this abundance of available HTTP servers demonstrates two things: first, that it is relatively easy to build a HTTP server, and second, that they will become a commodity very soon. Oracle expects HTTP to become an integral part of all new operating system releases, much like standard FTP or TELNET daemons.

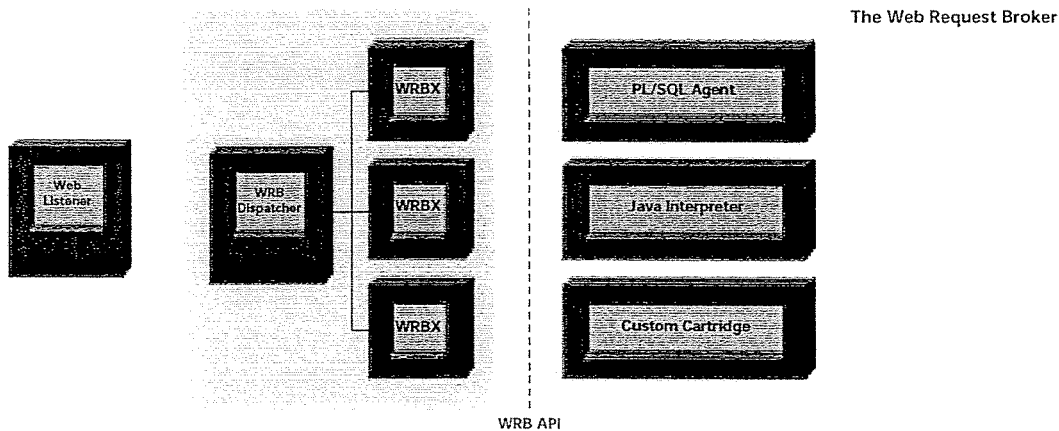
Oracle's mission is to provide an operating system independent development and runtime environment for business applications. As a direct consequence of this objective, we intend to provide support for an increasing number of HTTP engines. Initially, we will provide WRB adapters for the very popular Apache and NCSA servers, but also for other servers such as the Netscape Commerce Server and Open Market's WebServer. Note that these adapters are not provided in the initial release of WebServer 2.0, which only supports the Oracle Web Listener.

Integrating with any HTTP server



The Web Request Broker

Web clients connect to Oracle WebServer 2.0 using HTTP just like any other web server. A request is delivered in the form of a URL, which conceptually is an identifier for a web object (normally an HTML page). The HTTP engine does not make any attempt at interpreting the URL, instead it is immediately handed off to the Web Request Broker for further processing.



The WRB Dispatcher (WRBD)

The WRB dispatcher must decide what type of object is being requested. To do this, it examines the WRB configuration file, which maps virtual directories to WRB Services. For instance, the Web Request Broker may be configured to dispatch all URL requests beginning with /java to the integrated Java Interpreter. However, if no suitable WRB Service is found, the request is passed back to the Web Listener and standard processing continues. The WRB Dispatcher performs dynamic load balancing between multiple instances of a WRB Service, and the webmaster can configure each WRB Service to have a minimum and maximum number of instances. An instance corresponds to a process in a UNIX environment.

All inter-process communication is handled by the transport independent WRB protocol. WebServer 2.0 supports standard IPC mechanisms, and future releases will also support other mechanisms, such as for instance OMX, Oracle's CORBA 2.0 compliant Object Request Broker. The advantage of this is clear: WRB Services may be implemented as industry standard distributed objects, enabling true distribution and scalability across multiple nodes.

The WRB Execution Engine (WRBX)

Each WRB Service consists of a common WRB Execution Engine (WRBX) and a shared library, which is dynamically loaded at runtime. By providing an open API to WRBX, Oracle is encouraging partners and customers to integrate their own extensions. Integration is done at a much higher level than CGI or various HTTP server APIs.

The WRB API

The WRB API allows a developer to register three basic callback functions with WRBX: an initialization function, a request handler, and a shutdown function. The WRB API also contains numerous utility functions which may be invoked directly from the three basic callbacks.

The WRB API is being developed in cooperation with Oracle partners and customers, and we are using their feedback to finalize the specification and provide maximum openness and inter-operability between Oracle WebServer and third-party applications.

Building a Cartridge

This section will take you through the steps needed to build your own custom cartridge, or extension, to Oracle WebServer 2.0. Keep in mind that this application development platform is language independent—so you can use any programming language that your operating system supports, as long as your linker can build a shared library from the compiled code. This is a very powerful capability, which allows you to integrate existing legacy applications written in for instance COBOL, as well as develop new cartridges using C or C++. The steps involved in creating a new cartridge are as follows:

1. Create three functions: one to initialize the cartridge, one to shutdown, and one to handle a request.
2. Create an entry point function which takes one parameter of type pointer to a `WRBCallbacks` structure. Fill in the `WRBCallbacks` structure with your three functions.
3. Link your cartridge with the WRB library to produce a shared library (dynamic link library).
4. Register your library and entry point function in the WRB configuration file using the WebServer Manager.
5. Define one or more virtual directory mappings for your cartridge.

The full specification of the WRB API is available as part of the distribution. Please review it carefully before you attempt to build your first cartridge.

Oracle WebServer 2.0 ships with a sample cartridge called "Hello World!". On UNIX system, you can find the source code in the `ows2/sample/wrbsdk` directory. The file `hello.c` defines the three mandatory functions as follows:

```

WRBReturnCode test_init( WRBCTX, clientcxp )
void *WRBCTX;
void **clientcxp;
{
    return (WRB_DONE);
}

WRBReturnCode test_exec( WRBCTX, clientcxp )
void *WRBCTX;
void **clientcxp;
{
    WRBClientWrite(WRBCTX, "Content-type: text/html\n\nHello World\n", 40);
    return (WRB_DONE);
}

WRBReturnCode test_shut( WRBCTX, clientcxp )
void *WRBCTX;
void **clientcxp;
{
    return (WRB_DONE);
}

```

The first function, `test_init()`, will be our initialization function, which is called once when the cartridge is started. The third function, `test_shut()`, is the shutdown function, which is called once just before the cartridge is stopped. The final function, `test_exec()`, is the guts of the cartridge. It is invoked each time an incoming request is dispatched to this cartridge. As you have probably already guessed, this is a very simple cartridge: nothing is done in the initialization or shutdown functions, and the exec function simply prints a MIME header for an HTML document followed by the words "Hello World".

The next step is to register these three functions as callback function with the WRB Execution Engine (WRBX), which is the program that actually dynamically loads your cartridge at runtime. This is done in the function `testentry()`:

```

WRBReturnCode testentry (WRBCalls)
WRBCallbacks *WRBCalls;
{
    WRBCalls->init_WRBCallback = test_init;
    WRBCalls->exec_WRBCallback = test_exec;
    WRBCalls->shut_WRBCallback = test_shut;
    return (WRB_DONE);
}

```

Finally, you must compile and link your program to produce a shared library which can be loaded by the WRBX at runtime to instantiate your cartridge. The following shows how the shared library is built on Sun Solaris 2.4, but it will differ on just about every operating system:

```
$ cc -c -o helloworld.o -g -I${ORACLE_HOME}/ows2/wrbsdk/inc helloworld.c

$ cc -g -xs -L${ORACLE_HOME}/ows2/wrbsdk/lib -o helloworld.so -G helloworld.o
-lns1 -lm -lsocket -ldl -laio
```

Please use the Makefile included in the `hello` directory as a template for your operating system. Once the shared library has been successfully built, the next step is to register the cartridge with the Web Request Broker. You can do this with the WebServer Manager HTML user interface. The configuration file in the distribution already has the Hello World cartridge registered as follows (note that `/ora/7.3.2/` will be replaced by whatever your `ORACLE_HOME` is):

```
{Apps}

;

; APP      Object Path                                     Entry Point      Min      Max
; ===      =====                                     =====      ==      ==

OWA        /ora/7.3.2/ows2/lib/libndwoa.so                ndwoadinit       0        100
SSI        /ora/7.3.2/ows2/lib/ndwsssi.so                ndwssinit        0        100
JAVA       /ora/7.3.2/ows2/lib/libjava.so                ojsdinit         0        100
HELLO      /ora/7.3.2/ows2/sample/wrbsdk/ helloworld.so testentry          0        100

{AppDirs}

;

; Virtual Path      APP      Physical Path
; =====          ==      =====

/ssi                SSI      /ora/7.3.2/ows2/sample/ssi
/hr/owa             OWA      /ora/7.3.2/ows2/bin
/tr/owa             OWA      /ora/7.3.2/ows2/bin
/owa_dba/owa        OWA      /ora/7.3.2/ows2/bin
/java               JAVA     /ora/7.3.2/ows2/java
/sample/wrbsdk/hello HELLO   /ora/7.3.2/ows2
/sample/ssi         SSI      /ora/7.3.2/ows2/sample/ssi
/sample/java/run    JAVA     /ora/7.3.2/ows2/sample/java

{AppProtection}

/owa_dba/owa        Basic(Admin Server)

{SSI}

EnableLiveHTML      = TRUE
ParseHTMLExtn       = FALSE
EnableExecTag        = TRUE
ExtensionList        = html shml lhtml

{JAVA}

CLASSPATH            = /ora/7.3.2/ows2/java/classes.zip:/ora/7.3.2/ows2/java/oracle.zip
JavaCacheTimeout     = 86400
```

Each line in the [Apps] section is used to define one cartridge. The columns are:

- Cartridge name. A unique name assigned to the cartridge.
- Shared library name. The full path to your application.
- Entry-point. The name of the function that the WRBX will call to register your callbacks.
- The last two columns contain the minimum and maximum number of cartridge instances (threads or processes) that the Web Request Broker will start.

Each line in the [AppDirs] section defines a virtual directory path and a working directory for the cartridge. The columns are:

- Virtual directory. When the Web Request Broker sees a HTTP request for an object in this directory path, the request will be dispatched asynchronously to the corresponding cartridge (next column). Note that multiple paths may be defined for the same cartridge.
- Cartridge name. This must correspond to one of the cartridges defined in the [Apps] section.
- Working directory for the cartridge.

Each cartridge may optionally have its own section containing parameters, which are accessible through the WRB API. For instance, the Java Interpreter uses the `CLASSPATH` variable to determine where to look for Java classes.

That's all there is to it. This is the way Oracle provides cartridges to extend the base functionality of Oracle WebServer 2.0, and this is the way that 3rd parties and customers can do the same. You will notice that the Java Interpreter, PL/SQL Agent, and LiveHTML are implemented this way. Thus nothing prevents you from integrating any information management system, be it relational databases or legacy systems running on mainframes. If it can be accessed through any programming language on the WebServer node, it can be fully integrated through a custom cartridge and run as a WRB Service.

WebServer SDK

Oracle WebServer 2.0 includes two types of cartridges: system cartridges that perform a well defined function, and programmable cartridges that act as interpreters or runtime environments for applications. An example of a system cartridge is the Verifone VPOS (Virtual Point Of Sale) cartridge, which allows web applications to perform electronic payment transactions using a vast number of payment protocols over the Internet (Note: the VPOS cartridge is not included with Oracle WebServer 2.0, but you can get more information about it at <http://www.verifone.com>). Examples of programmable cartridges are the PL/SQL Agent, Java Interpreter, and LiveHTML, which are all covered in the following section.

The PL/SQL Agent

The PL/SQL Agent enables web applications to be developed using Oracle7 stored procedures written in PL/SQL. This is a tremendous advantage if you are building a data-driven application and your data happens to be stored in an Oracle7 database. Interacting with database objects is much easier in PL/SQL than any other language. It also has the benefit of being very scalable and completely portable between all operating systems supported by Oracle.

From a developer's perspective, the PL/SQL Agent is no different than the popular "Web Agent", which shipped with Oracle WebServer 1.0. Both enable developers to build data-driven, dynamic HTML pages with personalized content using Oracle7 stored procedures. Due to the new architecture in Oracle WebServer 2.0 with the Web Request Broker, the PL/SQL Agent is roughly an order of magnitude faster than the previous version. This is mainly due to the fact that each instance of the PL/SQL Agent stays connected to Oracle7 between requests, and thus does not need to establish a new database connection each time a stored procedure needs to be executed. Note that the Web Agent implemented through CGI is still bundled with WebServer 2.0 for backwards compatibility purposes, but its use is not recommended since the new architecture is so much faster.

The PL/SQL Agent performs two major tasks which enable a stored procedure to generate a dynamic HTML page. The first is to launch the procedure and pass in any parameters that were part of the HTTP request. The translation of a HTTP request to PL/SQL is not obvious, and we will therefore explore this in greater detail in a moment. The second task performed by the PL/SQL Agent is to provide a method for any stored procedure to actually deliver "output". In addition to performing these tasks, the PL/SQL Agent includes the PL/SQL Web Toolkit: a collection of packaged procedures, functions, and utilities to enhance productivity and ensure valid HTML output.

Launching a Procedure

For a request to be handed off to the PL/SQL Agent, the leading directory path must be found in the WRB configuration file. When you install WebServer 2.0, the following directory mappings are pre-defined:

/hr/owa	CWA	/ora/7.3.2/ows2/bin
/tr/owa	CWA	/ora/7.3.2/ows2/bin
/owa_dba/owa	CWA	/ora/7.3.2/ows2/bin

So any request starting with for instance the directory path `/tr/owa` will be recognized and handed off to the PL/SQL agent. Lets look at a sample URL:

```
http://www.oracle.com/tr/owa/tr.splash
```

This URL consists of three different parts: the first part is simply a pointer to the webserver (`http://www.oracle.com`). The second part is the directory path (`/tr/owa`) which has an entry in the WRB configuration file. At this point we know that we will be handing off the request to the PL/SQL Agent. What follows after the directory path (`/tr.splash`) is interpreted as the name of a

stored procedure (the leading slash is ignored), so in this example the PL/SQL Agent will attempt to run a procedure named `tr.splash`. The actual procedure is named `splash` and is part of the public interface to a package named `tr`, hence the dot-notation.

Connecting to the Database

The Oracle7 server has a higher level of security than flat files managed by an operating system. Users are authenticated using a wide range of mechanisms ranging from simple passwords to fingerprint scanners, smart-cards, and distributed protocols such as DCE and Kerberos. The PL/SQL Agent also needs to authenticate itself to gain access to a distinct schema of objects and storage space within the database. It also needs to have access to all the details required to actually establish a permanent connection: typically a set of environment variables (`ORACLE_HOME` and `ORACLE_SID` for UNIX systems) and perhaps a SQL*Net connect descriptor. All of this information is referred to as a DCD (Database Connection Descriptor), and all of the DCDs known to the PL/SQL Agent are stored in a configuration file named `owa.cfg` (DCDs were known as Web Agent Services in WebServer 1.0). Care should be taken to prevent unauthorized access to this file, since it contains sufficient information to gain access to the database. DCDs are defined in the PL/SQL Agent's configuration file, but they are activated by creating a directory mapping in the Web Request Broker's configuration file.

At the lowest level, Oracle's internal API allows us to establish a connection to an Oracle7 Server using SQL*Net in one step, and actually logging on to the database in a separate step. Since each HTTP request needs a new session, we are technically logging on and off between requests, but this is still extremely fast since the Oracle7 Server connection is maintained between requests.

The token immediately preceding the keyword `owa` in the directory path is interpreted as the name of a DCD. For instance, `/tr/owa` would cause the PL/SQL Agent to look for a DCD named `tr`. If such a DCD does not exist, an attempt is made to use the default DCD. If no procedure is found or if the default DCD has not been created, the request will fail.

In Oracle WebServer 2.0 one or more instances of the PL/SQL Agent will be running for each active DCD. Incoming requests will share the same DCD if they are using the same directory path, and authentication of the user will be handled at the HTTP level by the Oracle Web Listener. You will typically configure the Oracle WebServer to have at least one public DCD and several protected DCDs requiring user authentication. When a user has authenticated with a username and password, the username is available to the stored procedure being run, but it is up to the application developer to implement security.

Passing Parameters

The HTTP protocol supports several methods of passing information from the client to the server along with a request. The POST method passes parameters in a stream (standard input) directly to the Web Request Broker (or a CGI program). The GET method tags parameters onto the end of a request. For instance, if two parameters were to be passed to the `tr.splash` procedure, the complete URL would look like this:

```
http://www.oracle.com/tr/owa/tr.splash?name1=value1&name2=value2.
```

The parameters are encoded and passed as name/value pairs with `&` as a delimiter. If you were developing a web application from scratch, you would have to parse this "query string" and decode the parameters yourself. Fortunately, the Web Request Broker does this automatically for all cartridges, and the PL/SQL Agent converts the parameters to PL/SQL procedure parameters. Furthermore, you don't have to care about what method is used to pass the parameters, they are simply delivered as parameters to your procedure. To complete this example, the URL above with the two parameters would be translated into a PL/SQL procedure call like this:

```
begin
    tr.splash ( name1 => 'value1', name2 => 'value2' );
end;
```

This parameter passing notation may not be familiar to you even if you are an experienced PL/SQL developer, but naming each parameter this way essentially enables us to not worry about the order they appear in. For instance, this procedure could actually have been defined like this:

```
create package tr as
    procedure splash ( name2 in varchar2 := null, name1 in varchar2 := null);
end;
```

Note that all parameters have default values (`null`). This ensures that the procedure will be found even if one or more parameters are missing. One additional feature is worth mentioning: if you have a large or variable number of parameters, the PL/SQL Agent allows you to use PL/SQL tables of `varchar2` as input parameters. Consider an electronic storefront. The user will first search through the product catalog "putting" items into an electronic shopping cart. When the order is ready to be processed, an HTML purchase form with a variable number of line-items needs to be submitted from the browser. If the input fields for each line-item have identical names in the HTML form (e.g. re-occurring `item_partno` and `item_quantity`), the following procedure could handle the request:

```
create package shop as
    type vc2arr is table of varchar2(255) index by binary_integer;
    procedure buy ( item_partno in vc2arr, item_quantity in vc2arr);
end;
```

We have now covered the basics of how an Oracle7 stored procedure is launched by the PL/SQL Agent and how input parameters are passed from the client browser to the procedure. The next section explains how to generate the resulting dynamic HTML using the PL/SQL Web Toolkit.

Generating HTML with the PL/SQL Web Toolkit

The toolkit consists of a number of packages that are installed into each schema corresponding to a DCD when the DCD first is created. The two packages named `HTP` and `HTF` (HyperText Procedures and HyperText Functions) make up the core of the PL/SQL Web Toolkit and provide a low-level one-to-one mapping between HTML 2.0 and PL/SQL procedures and function. At the lowest level, the procedure `http.print` plays a central role, in that it is ultimately used either directly by the developer or indirectly by a higher-level procedure or function to put something into the result, which is transmitted back to the client. Although the result will usually be a HTML document, any MIME type is supported, including user-defined types. Here is an example of a simple procedure:

```
create procedure hello is
begin
    http.print('Hello World');
end;
```

The `http.print` procedure simply puts whatever parameter you pass to it on a page. A shortcut procedure is available with the name `http.p`. You can of course add HTML tags manually, or you can use the functions and procedures in the PL/SQL web toolkit that encapsulate HTML for you. The following two procedures generate the exact same result, `hello1` simply prints the result exactly as desired, whereas `hello2` uses the toolkit procedures and functions:

```
create procedure hello1 is
begin
    http.p('<title>Hello Sample</title>');
    http.p('<h1>Hello Sample</h1>');
    http.p('Hello World, this is <b>easy</b>');
    http.p('<hr>');
    http.p('<a href="http://www.oracle.com">Visit Oracle on the Web!</a>');
end;

create procedure hello2 is
begin
    http.htitle('Hello Sample');
    http.p('Hello World, this is ' || htf.bold('easy'));
    http.line;
    http.anchor('http://www.oracle.com', 'Visit Oracle on the Web!');
end;
```

The second example demonstrates a powerful way of bundling multiple HTML tags into widgets. The `http.htitle` procedure generates both a title and a header tag. This concept can easily be expanded to higher levels, so you could

for instance define a common procedure to set your document title and header, background bitmap, text and link colors, and navigation toolbar. In addition to increasing productivity, this technique automatically brings a consistent look and feel to your applications.

Java Interpreter

For a general introduction to Java, please visit <http://java.sun.com>. This section assumes general knowledge of Java as a technology, although a very brief introduction follows. Java is an objective-oriented language strongly resembling C++ but with a few restrictions that strongly improve application robustness, the most important being no explicit pointer manipulation and no multiple class-inheritance.

Java allows you to develop three types of classes: normal Java classes which are used as building blocks or abstract classes, runnable classes, known as Java applets, which can be downloaded by a Java-enabled web browser, and Java applications, which are standalone programs. Oracle WebServer 2.0 allows you to integrate the last type, standalone applications, into your website using the Oracle Java Interpreter. Oracle WebServer 2.0 (and 1.0 for that matter) can also act as a repository for Java applets to be downloaded to clients, but that is a different topic not covered here.

The distinguishing attribute of a Java application compared to an applet or other class, is that it contains a public method (function) called `main`. This method acts as the entry-point to the application. Oracle WebServer 2.0 ships with a few sample Java applications, of which the simplest looks like this:

```
class HelloWorld {
    public static void main (String args[] ) {
        System.out.println("Content-type: text/html\n\n");
        System.out.println("Hello World!");
        System.out.println("Hello World!");
    }
}
```

Anything printed to the standard output stream from a Java application is simply delivered back to the client.

Extending Java

Oracle WebServer also contains the Oracle Java Web Toolkit, three Java packages to extend the capabilities of the standard Java interpreter. These are `oracle.html.*`, which contains objects for dynamic HTML generation, `oracle.rdbms.*`, which contains objects for database access, and `oracle.plsql.*`, which contains objects for PL/SQL access. These packages must be explicitly imported by any application that intends to use them.

Executing PL/SQL from Java

The Java Interpreter interfaces to the Oracle7 Server by running PL/SQL packages or standalone PL/SQL procedures and functions. Each package or procedure must have a Java package wrapper, which is a Java class containing methods to call that package's procedures and functions. Standalone procedures and functions are all wrapped in a single wrapper. Once you have identified or created the PL/SQL packages your applications needs, you can create the package wrappers automatically by running the `p12java` utility as follows:

```
p12java {flags} username/password[@connect-string] packagename...
```

This utility creates a Java wrapper class for each package or procedure given as an argument to the command. When your application is run, it creates an instance of this class to interface to the package. The following is a summary of steps required to write a Java application which interacts with an Oracle7 Server through PL/SQL:

1. Create an object of type `Session` to handle the database connection. All of the operations performed during this session must be called from within this object.
2. For each PL/SQL package used in the application, you must create one instance of the `packagewrapper` subtype created for that PL/SQL package by the `p12java` utility.
3. For each parameter of a PL/SQL package, you must create an instance of the Java variable that matches that parameter.

The following example uses a packaged function named `employee.count_employees` to get a tally of people in a given department. For the sake of simplicity, all error and exception handling has been left out.

```
import oracle.rdbms.*;
import oracle.plsql.*;

public class EmployeeReport {
    public static void main (String args[]) {
        .
        .
        int tally = get_count("SALES");
        .
        .
    }

    int get_count( String deptName ) {
        Session.setProperty("ORACLE_HOME", "/user/oracle");
        Session.setProperty("TNS_ADMIN", "/user/oracle/network/admin");

        Session session;
        session = new Session("scott", "tiger", "HR_DB");

        // Create a new instance of Employee package
        Employee employee = new Employee(session);

        // Create object to encapsulate PL/SQL parameter
        PDouble      pEmployeeCount;
```

```

// ADD: call Employee package to count the number of
// employees in the department
pEmployeeCount = employee.count_employees(pDeptName);
return (int)pEmployeeCount.doubleValue();
}

```

Generating HTML from Java

A Java application can generate a dynamic HTML page. Essentially, anything written to standard output by a Java application run by the Java Interpreter will be returned to the client browser. The "Hello World" example described earlier demonstrates this. Similar to the PL/SQL Web Toolkit, the Oracle Java Web Toolkit allows you to generate HTML without having to hardcode the actual HTML tags.

To generate dynamic HTML from within Java, you create various objects that use the interface `IHtmlItem`. All classes that generate dynamic HTML implement this interface, which has two simple methods: `toHTML` and `print`. Both of these methods produce the content of the object as HTML, but `toHTML` returns it as a string, whereas `print` sends it to system output. Therefore, you use `toHTML` when you want the resulting HTML to be further processed by another method and use `print` when you want to output it. In effect, you build your Web page in a buffer with `toHTML` and flush the buffer with `print`.

The `oracle.html` package provides a standard set of classes based on HTML2, HTML3, and popular browser-specific extensions. You are not limited to these, however. You can easily create your own customizable HTML classes by deriving them from the `CompoundItem` or `Container` classes. The `oracle.html` package also has the intelligence to generate output that is optimized for the browser at hand. For example, a browser that does not support tables will get table data in the form of pre-formatted strings. Here is a simple example:

```

import oracle.html.*;

public class HelloWorld {
    public static void main (String args[]) {

        // Create an HtmlHead Object titled "Hello World!"
        HtmlHead hd = new HtmlHead("Hello World!");

        // Create an HtmlBody Object
        HtmlBody bd = new HtmlBody();

        // Create an HtmlPage Object
        HtmlPage hp = new HtmlPage(hd, bd);

        // Adds a simple string "Hello World" in this page
        bd.addItem("Hello World!");

        // Print out the content of this Page
        hp.print();
    }
}

```

LiveHTML

One of the most popular features of the Web is known as *Server Side Includes* (SSI). Certain documents can be parsed by a web server before sending them back to the client, and they may contain "include" directives instructing the web server to insert information into the document on the fly. Examples of use are page-hit counters, timestamps, etc. Server Side Includes literally bring otherwise static HTML pages to life, hence the name LiveHTML. Oracle WebServer 2.0 supports SSI+, a common extension to NCSA's original specification.

LiveHTML code is formatted as SGML comments, so that it is ignored should the file ever find its way to the browser unparsed. The format for LiveHTML codes, therefore, is the following:

```
<!--#command tag1="value1" tag2="value2" -->
```

The tags are arguments to the commands, most of which actually only accept one of the possible tags. The possible commands and their associated tags are as follows:

config. This command sets parameters for how the file or script is to be parsed and therefore is normally the first LiveHTML command in a file. The possible tags are:

- **errmsg.** This specifies the error message that is sent to the client if an error occurs while parsing this document. Here is an example:

```
<!--#config errmsg="A parse error occurred in the Hockey_Fass file"-->
```

- **timefmt.** This specifies a date format. LiveHTML files frequently include timestamps. The conventions follow the strftime library call supported in most versions of Unix, even if the WebServer is not running on a Unix platform.
- **sizefmt.** This specifies the format used when displaying a file size. The possibilities are bytes, which gives the absolute size in bytes, and abbrev, which gives the size in kilobytes or megabytes as appropriate.
- **cmdecho.** This specifies whether non-CGI scripts subsequently executed have their output incorporated into this HTML page. The possible values are ON and OFF. ON specifies that the output is included. The default is OFF.
- **cmdprefix.** Specifies a string that will be prepended to each line of the script output.
- **cmdpostfix.** Specifies a string that will be appended to each line of the script output.

include. This command specifies that a file is to be included in the generated HTML page at this point. The file can any of the following:

- another LiveHTML file like the current one.

- a regular HTML file.
- an ASCII file, which of these it is determined, as usual, by the extension.

The possible tags are:

- **virtual**. This gives a virtual path to the file. The directory mappings for virtual paths are set by the WebServer administrator using WebServer Manager.
- **file**. This gives a pathname relative to the current directory. References to parent directories or uses of absolute pathnames are forbidden.

echo. This gives the value of an environment variable. This variable is either one of the standard CGI environment variables or one of the LiveHTML extensions, which currently are all standard Server Side Include variables. There is only one tag, **var**, and it must be present. It provides the name of the variable. The LiveHTML environment variables are as follows:

- **DOCUMENT_NAME**. The current filename.
- **DOCUMENT_URL**. The virtual path to this file.
- **QUERY_STRING_UNESCAPED**. If the client sent a query string, this is an unescaped version of it, with all shell-special characters escaped with \.
- **DATE_LOCAL**. The current date and local time zone in the format specified by the most recent config timefmt command.
- **DATE_GMT**. Same as the above, but in Greenwich mean time.
- **LAST_MODIFIED**. The last modification date of the file, given in the format specified in the last config timefmt command.
- **fsize**. This produces the size of the file in the format specified in the most recent config filesize command. Tags are the same as for include.
- **flastmod**. This produces the last modification date of the file in the format specified in the most recent config timefmt command. Tags are the same as for include.
- **exec**. This is the command to execute a script. The tags specify whether or not the script is CGI.
- **cmd**. This specifies a non-CGI script. Execution is passed to the Operating System, and the given string is parsed as though it were entered at a command-line interface. The full path of the script must be given. The non-CGI environment variables specified under echo above can be referenced. Whether the output of the script is included in the HTML page that the parser outputs, is determined by the most recently executed config cmdecho.
- **cgi**. This specifies a CGI script. The value given will be the virtual path of the script. URL locations are automatically converted into HTML anchors.

Summary

First generation web servers have fundamental shortcomings in the areas of scalability, management, and extensibility. Oracle is leveraging its core competency in these areas to deliver a next-generation product capable of addressing these issues and making extremely large websites with hundreds of thousands of users and terabytes of information a reality. Oracle WebServer 2.0 lays the foundation and provides a first glimpse of new emerging technology, which will be fully developed in forthcoming releases. After all, this is Internet Technology, and we are finding our product life-cycles being reduced from years to months.

Oracle WebServer 3.0 is scheduled for release in late 1996, and we expect the Web Request Broker API to be fully developed by that time. Oracle is actively working with partners and developers to provide a solid platform for application development. Other plans for WebServer 3.0 include making extensive use of Oracle7 for website configuration management, including user authentication, optionally storing content with automatic replication to multiple sites, more cartridges, inter-cartridge processing, and full support for all major HTTP listeners. It is going to be an exciting year—be sure to keep an eye on our public website, <http://www.oracle.com>, for upcoming news, releases, and events.

Magnus Lonnroth
Senior Product Manager
Oracle Internet Server products

Terminology and Acronyms

Term	Description
Apache	An increasingly popular HTTP server on the Internet
API	Application Programming Interface
CGI	Common Gateway Interface
daemon	Detached process waiting for incoming service-requests over the network
FTP	File Transfer Protocol
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
Java	An object-oriented language for the Web
JavaScript	A language developed by Netscape for use in the Netscape Navigator and the Netscape Commerce Server through NSAPI
JDBC	ODBC for Java
LiveHTML	Currently same as Server Side Includes.
MIME	Multi-purpose Internet Mail Extensions
NCSA	National Center for Super-computing Applications
NSAPI	Netscape Server API
PL/SQL	Oracle's procedural language for stored procedures
SNMP	Simple Network Management Protocol
SQL	Structured Query Language
SQL*Net	Oracle's network protocol which runs on top of most networks
SSI	Server Side Includes
SSL	Secure Sockets Layer
TELNET	Terminal Emulation Program
URL	Uniform Resource Location
VRML	Virtual Reality Markup Language
W3C	World Wide Web Consortium
WRB	Web Request Broker
WRB API	An API for developing custom cartridges
WRBD	Web Request Broker Dispatcher
WRBX	Web Request Broker Execution Engine

Exhibit 40

Dienst: Implementation Reference Manual

Carl Lagoze* Erin Shaw† James R. Davis‡ Dean B. Krafft§

May 5, 1995

Abstract

We describe the architecture and implementation of Dienst: a protocol and server that provides distributed document libraries over the World Wide Web. Dienst is based on a document model that incorporates unique document names, multiple document formats, and multiple document decompositions. Interoperability among Dienst servers provides the user with a single logical document collection, even though the actual collection is distributed across multiple servers. The Dienst protocol uses HTTP (the protocol of the World Wide Web) as a transport layer, making Dienst servers accessible from any WWW client. Dienst is currently used as the infrastructure for a distributed computer science technical report library by a number of U.S. universities. This document is intended as a guide for Dienst site administrators and implementors of other digital library systems. It describes the architecture of an individual server and network of Dienst servers and includes the server installation instructions. Appendices describe copyright issues and retrospective conversion of the Cornell technical report collection.

*Computer Science Department, Cornell University, Ithaca, NY 14853, lagoze@cs.cornell.edu

†Computer Science Department, Cornell University, Ithaca, NY 14853, shaw@cs.cornell.edu

‡Xerox Corporation, Design Research Institute, Cornell University, Ithaca, NY 14853, davis@dri.cornell.edu

§Computer Science Department, Cornell University, Ithaca, NY 14853, dean@cs.cornell.edu

Contents

1	Overview of Dienst	5
2	Organization of this document	7
3	Dienst document model	7
3.1	Unique document names	7
3.2	Multiple document formats	9
3.3	Document decompositions	9
4	Dienst server architecture	9
4.1	Components of a Dienst server	9
4.2	Interaction among Dienst servers	11
5	Dienst user interface features	12
6	Installing a basic Dienst system	21
6.1	Considering copyright issues	21
6.2	Obtaining publisher IDs for your site	21
6.3	Creating your document database	21
6.3.1	Set up a directory structure	22
6.3.2	Make the bibliography files	22
6.4	Downloading the Dienst server	24
6.5	Configuring your server	25
6.5.1	Modifying config_constants.pl	26
6.5.2	Writing code in custom.pl	26
6.5.3	Declaring storage formats in custom.pl	28
6.6	Checking the configuration and database	29
6.7	Indexing your document database	29
6.8	Starting your server	29
6.9	Testing your server	30
6.10	Registering your site	30
6.11	Downloading a Web server	31
6.12	Configuring your Web server to work with Dienst	31
6.12.1	Creating the home page	31
6.12.2	Configure NCSA httpd	32
6.12.3	Configure CERN httpd	32
6.13	Testing your Web server with Dienst	33
7	Adding additional document types to your Dienst server	33
7.1	Overview of document types	33
7.2	Adding new types to custom.pl	33
7.3	Adding new file formats to documents in your database	35

8	Installing Dienst Optional Features	35
8.1	Installing the Dienst Full-Text Search Package	35
8.1.1	Overview of full-text search	35
8.1.2	Prerequisites for this installation	36
8.1.3	Overview of this installation	36
8.1.4	Detailed installation instructions	36
8.1.5	Additional information	38
8.2	Installing the Dienst Page-Level Zoom Package	38
8.2.1	Overview of zoom	39
8.2.2	Prerequisites for this installation	39
8.2.3	Overview of this installation	39
8.2.4	Detailed installation instructions	39
8.3	Installing the Dienst Foreign Servers Package	41
8.3.1	Overview of foreign servers	41
8.3.2	Prerequisites for this installation	41
8.3.3	Overview of this installation	41
8.3.4	Detailed installation instructions	42
8.4	Installing the Dienst Printing Package	42
8.4.1	Overview of printing and downloading	42
8.4.2	Prerequisites for this installation	42
8.4.3	Overview of this installation	43
8.4.4	Detailed installation instructions	43
8.5	Maintaining your Dienst server	44
8.5.1	Adding new documents to your database	44
8.5.2	Monitoring log files	45
8.5.3	Maintaining the integrity of your database	45
9	The Dienst Library Management Package	47
9.1	Submission package overview	48
9.2	Format generator utility	49
9.3	Library management tools overview	49
9.4	Notes on directory structure and permissions	51
9.5	Prerequisites for this installation	51
9.6	Configuring the submission package	52
9.7	Configuring your Web server to enable Submissions	52
10	Advanced server customization	53
10.1	Replacing the Dienst built-in search engine	53
10.2	Using other bibliography formats	55
A	Dienst protocol	57
A.1	Repository requests	57
A.1.1	Requests about the entire collection	57
A.1.2	Requests about an individual document	58
A.2	Indexer requests	58
A.3	User interface requests	60
A.4	Miscellaneous Requests	61

B	Copyright considerations	62
B.1	Copyright handling at Cornell	62
B.1.1	Cornell copyright agreement form	63
B.2	Copyright handling at Stanford	64
B.2.1	Stanford single report copyright permission form	64
B.2.2	Stanford blanket copyright permission form	65
C	Building the Cornell collection	67

server registration - As described earlier, each docid that identifies a document in a Dienst collection consists of a publisher identifier and a name. Each server in a set of interoperating Dienst servers indexes and acts as a repository for documents from one or more publishers. In the current version, all the documents for a publisher must be indexed and reside on the same server. Each server is able to locate the indexing and repository site for a specific publisher using tables that are maintained by a distinguished server, the meta server, and periodically downloaded by the individual server. This allows every server to provide the next two functions.

distributed searching - Each Dienst user interface server provides a front-end for searching the collection provided by all inter-operating Dienst servers. The user specifies in the search form which publishers should be searched. The respective user interface server then dispatches search requests, in parallel, to the Dienst index servers that correspond to the publisher selection(s). The user interface server then parses and combines the results from the requested servers to present a coordinated “hit list” to the user.

distributed document access - Each server, in a set of interoperating Dienst servers, is capable of responding to a request for any document in the distributed collection. The respective server uses the publisher in the docid of the requested document and the mapping tables downloaded from the meta server to route the document request to the server that acts as the repository for the publisher. This routing is transparent to the user.

5 Dienst user interface features

Each Dienst server allows the user to search, browse, and view the collection using any Web client. Several of the features of the user interface are described in this section. Not all Dienst servers provide all the features described in this section, nor are the features available for all documents.

- A number of features are optional (e.g. full-text searching) and may not be installed at the respective server.
- A number of features are dependent on the format(s) in which the respective document is available. For example, in-line page viewing is only available if the document is stored in GIF format.

Forms-based distributed searching - An HTML form allows a user to specify criteria for a search. These criteria are publisher name(s) (at least one must be selected), document name, and bibliographic keywords (title, author, abstract). The search form allows the user to specify *and* or *or* boolean operators between the bibliographic keyword fields. In addition, the user may use boolean operators and precedence symbols (parentheses) within bibliographic fields. An example search form is shown in figure 3.

Unified hypertext hit list - The results of a search (that may have been dispatched to several servers) are presented to the user as a combined list of hyperlinks sorted by publisher and docid. Each *hit* is link to the document summary page for the respective document. If any of the servers needed for a search (as specified in the publisher list in the search form) are not available, this information is printed at the top of the results form. An example hit list is shown in figure 4.

Search the Collection

Select one or more publishers from this list:

Princeton CS	▲
Stanford	▼
U Maryland College Park CS	▼

or ☐ search all publishers

Document Identifier:

Bibliographic keywords: (▲ AND keyword fields ▼ OR keyword fields)

Author:

Title

Abstract

☐ submit search to other technical report indexes

Figure 3: The Dienst search form allows users to enter search criteria for bibliographic fields. Boolean expressions may be used within and/or between fields.

Your search was for

publishers selected:

- Cornell
- Stanford
- U Maryland College Park CS

Bibliographic keywords ("and"ed together)

- author = subramanian or saltz or wei
- abstract = parallel algorithm

Search results

- [CORNELLCS:TR92-1294 *A Singular Loop Transformation Framework Based on Non-Singular Matrices*. Wei Li and Keshav Pingali.](#)
- [CORNELLCS:TR94-1469 *COMPILING FOR NUMA PARALLEL MACHINES*. Wei Li.](#)
- [STAN:CS-TR-89-1275 *A new approach to stable matching problems*. Ashok Subramanian.](#)
- [STAN:CS-TR-89-1278 *The complexity of circuit value and network stability*. Ernst W. Mayr and Ashok Subramanian.](#)
- [UMCP-CSD:CS-TR-3425 *Parallel Monte Carlo Simulation of Three-Dimensional Flow over a Flat Plate*. Robert P. Nance, H. A. Hassan, Richard G. Wilmoth, Bongki Moon and Joel Saltz.](#)
- [UMCP-CSD:CS-TR-3447 *Interprocedural Compilation of Irregular Applications for Distributed Memory Machines*. Gagan Agrawal and Joel Saltz.](#)

Figure 4: Dienst search results are displayed in a hypertext list, sorted by publisher and *docid*. Each *hit* is a link to the summary page for the respective document.

Document summary page - The document summary page serves two purposes. First, it displays the metadata associated with an individual document (e.g. author, title, publication date, abstract). If the search criteria for a document included abstract words, the appropriate abstract words are highlighted in the document summary page. Second, it presents to the user the formats in which the document is available. These are divided into four groups:

- page thumbnails for browsing.
- a structural overview of the document.
- formats for which physical page decompositions are available; the user may choose the appropriate page to display.
- formats for which decompositions are not available; selection of one of these formats will effect a download of the entire document in the respective format.
- a link to the printing and downloading form.

Only those formats available for the specific document will be shown on that document's summary page. An example document summary form is shown in figure 5.

A Singular Loop Transformation Framework Based on Non-Singular Matrices

CORNELLCS TR92-1294

Wei Li and Keshav Pingali
July 1992

In this paper, we discuss a loop transformation framework that is based on integer non-singular matrices. The transformations included in this framework are called λ -transformations and include permutation, skewing and reversal, as well as a transformation called loop scaling. This framework is more general than the existing ones; however, it is also more difficult to generate code in our framework. This paper shows how integer lattice theory can be used to generate efficient code. An added advantage of our framework over existing ones is that there is a simple completion **algorithm** which, given a partial transformation matrix, produces a full transformation matrix that satisfies all dependences. This completion procedure has applications in **parallelization** and in the generation of code for NUMA machines.

How to view this document

- [Display an overview of thumbnail pages.](#)
- Display a selected page in one of the following formats (document has 24 pages).

inline gif image	1	Display page
raw OCR output		
hi-resolution tiff image		
- Display the whole document in one of the following formats.
 - ☐ OCR text (produced by OCR, may have errors) 39910 bytes.
 - ☐ PostScript 90795 bytes.
- [Print or download all or selected pages.](#)

Figure 5: The document summary page displays the metadata associated with the respective document and the formats in which the document is available. The user can use the links on this page to display the document in the chosen format.

Structural overview - A structural overview allows the user to view the major structural features of a document, such as chapters, sections, and the like. An example structural overview is shown in figure 6.

Tools for Monitoring and Controlling Distributed Applications.

Keith Marzullo and Mark D. Wood

January 1991

<u>Title page</u>	1
<u>1 Constructing Reactive Systems</u>	3
<u>2 The Meta Architecture</u>	4
<u>3 Application Instrumentation</u>	5
<u>3:1 Access to Base Values</u>	6
<u>3:2 Functional Composition</u>	7
<u>3:3 Aggregates</u>	7
<u>3:4 Fault-Tolerance</u>	7
<u>4 Control</u>	8
<u>4:1 Interpreting Guarded Commands</u>	8
<u>4:2 Atomic Guarded Commands</u>	9
<u>4:3 Example</u>	10
<u>5 Discussion</u>	12
<u>5:1 Related Work</u>	12
<u>5:2 The ISIS System</u>	12
<u>5:3 Status</u>	13
<u>5:4 Directions</u>	14

Figure 6: The structural overview allows the user to see the hierarchical structure of the document, revealing features such as chapters and sections. Each structural item is a link to the page on which the item begins.

Page browsing via thumbnail images - Page thumbnails allow users to quickly browse the body of a document and discover and locate structural objects such as tables, graphs, and the like. Each thumbnail page is implemented using a HTML *imagemap*, allowing the user to select on a page and dispatch a URL to view that specific page in readable form. An example thumbnail page is shown in figure 7.

A Singular Loop Transformation Framework Based on Non-Singular Matrices

Wei Li and Keshav Pingali
July 1992

Section 1 of 1. Click within a page rectangle to display a full size page.

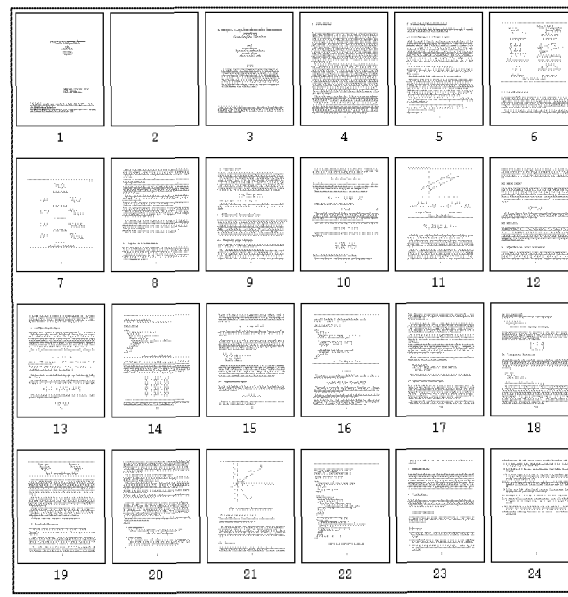


Figure 7: Page thumbnails are a tool for quick browsing of a document for formatting features (e.g., tables). The thumbnails are implemented using the HTML *imagemap* mechanism, which allows a user to select a page and view that page in full.

Inline page image - Inline page images are displayed in a resolution sufficient for on-screen reading of the document. At Cornell we have found that 72 dpi, four-bit deep GIFs have the proper combination of resolution and size for most monitors. The page image display also includes several other user interface objects:

- hyperlinks that allow the user to go the next or previous page of the document, or to the document summary page.
- two radio buttons that control the operation that is performed when the user selects a point within the document page. The two operations are zoom and full-text search, both of which are described below.

An example page-image page is shown in figure 8.

Page level zooming - The user can zoom in on a section of a page image by selecting the *zoom* radio button and clicking within the page image in the page-image display. If the document

[Prev](#) [Next](#) [Summary](#) [Page](#)

Click with the mouse to select a paragraph or section of the page.

🔍 Full-text search on a paragraph. 🔍 Zoom-in on a section.

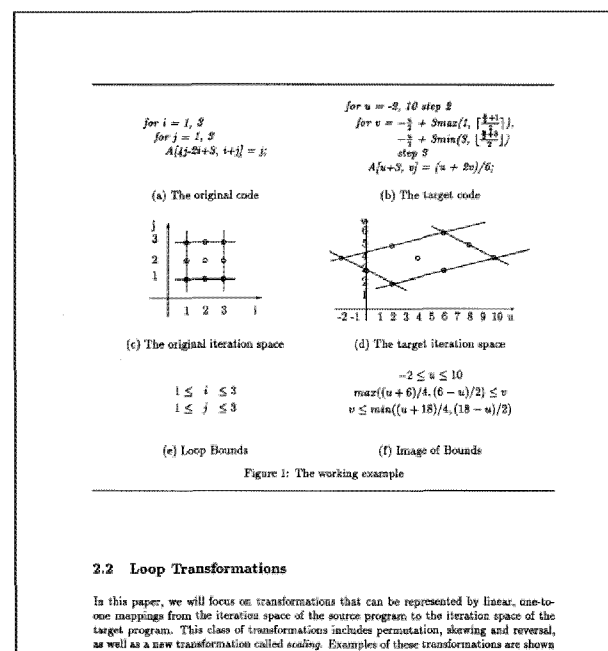
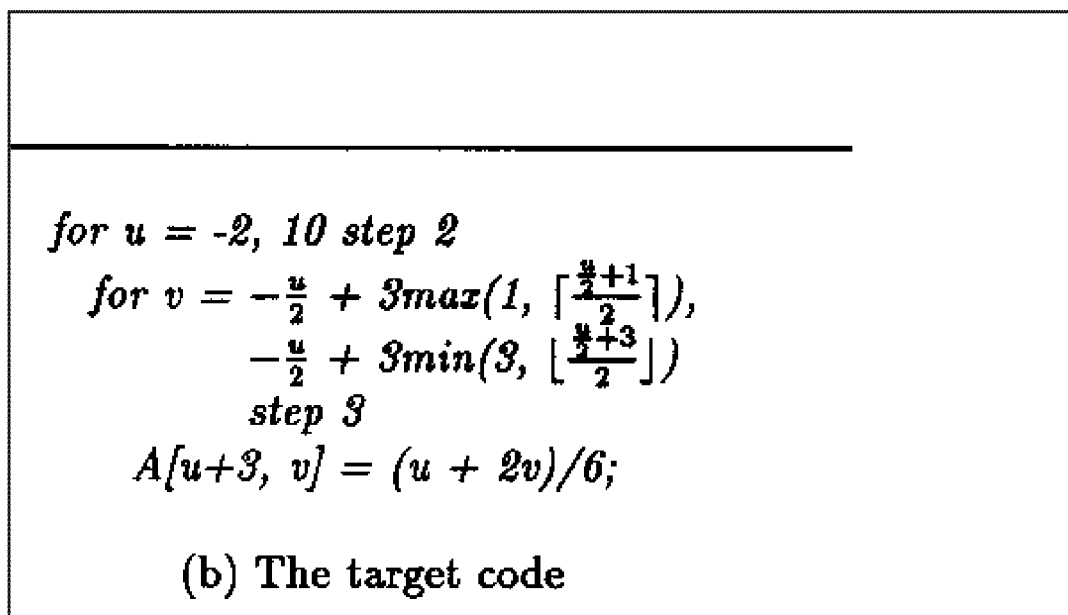


Figure 8: Pages are displayed in 72dpi GIF format with 4-bit deep gray scaling for maximum screen readability. The page display includes links to move back and forth through the document and controls for the zoom and *click-to-search* full-text search feature.

is available in a high-resolution format (e.g. scanned TIFF), a zoomed section of this format will be displayed to the user. An example zoom page is shown in figure 9.

Left Up Down Zoom Out



Left Up Down Zoom Out

Figure 9: Page level zooming allows the user to see fine detail on a document page.

Click-to-search full-text search - The user can perform a full-text search based on text in the document by selecting the *full-text* radio button (the default) and clicking within the page image in the page-image display. The server will use the contents of the paragraph in the proximity of the user selection as the source for the search. Mapping from the selection point to the ASCII text is done by an analysis of the OCR output using paragraph delineation heuristics (based on line spacing). A full description of the implementation of full-text searching is in section 8.1. A full-text search returns two sets of results: document level in ranked order and paragraph level in ranked order. The results are presented as hyperlinks so the user may then dispatch a request for the specific document. Note that full-text search in the current release of Dienst only searches the collection on the server to which the search has been submitted. This is because not all servers implement full-text search, and the ranking of distributed full-text searches is problematic. An example result page from a full-text search is shown in figure 10.

Printing and downloading - Users may choose to print or download all or selected pages from a document. The site administrator decides at server installation times the net domains from which a client request must originate for printing to be available to that client. In addition the administrator decides which printers are available to specific client domains. Clients outside the “printable” domains receive only a form that allows them to download all or selected pages of the document. In the current implementation printing is allowed for documents that are in

Full Text Search Results

Search Text

Figure 1: The working example 2.2 Loop Transformations In this paper, we will focus on transformations that can be represented by linear, one-to-one mappings from the iteration space of the source program to the iteration space of the t& program. This class of transformations includes permutation, skewing and reversal, as well as a new transformation called scaling. Examples of these transformations are shown in Figure 2. These transformations are standard except for scaling which corresponds to replacing a loop iteration variable by an integer multiple of it.

Search Options Selected

- show document links
- show paragraph links

NOTE: For most TRs, full text is generated by running OCR on the scanned image. Depending on the quality of the original, this may result in some errors in the paragraph text and may also affect the results of the search.

Full documents in relevance order.

- 0.45 [CORNELLCS:TR92-1294](#)
A Singular Loop Transformation Framework Based on Non-Singular Matrices. [Wei Li](#) and [Keshav Pingali](#).
- 0.31 [CORNELLCS:TR94-1434](#)
On Program Transformations. [Sofoklis G. Efremidis](#).
- 0.31 [CORNELLCS:TR93-1389](#)
An Algorithm for Processing Program Transformations. [Sofoklis G. Efremidis](#) and [David Gries](#).
- 0.30 [CORNELLCS:TR88-922](#)
Compaction-Based Parallelization. [Alexander Aiken](#).
- 0.28 [CORNELLCS:TR92-1278](#)
Access Normalization: Loop Restructuring for NUMA Compilers. [Wei Li](#) and [Keshav Pingali](#).
- 0.26 [CORNELLCS:TR85-678](#)
Percolation Scheduling: A Parallel Compilation Technique. [Alexandru Nicolau](#).
- 0.26 [CORNELLCS:TR86-792](#)
A Fine-Grain Parallelizing Compiler. [Alexandru Nicolau](#).

Figure 10: The *click-to-search* feature allows a user to select a paragraph on a page image as the input for a full-text search. Both document-level and paragraph-level results are returned from a full-text search.

PostScript or TIFF format. Documents in TIFF format are translated to ASCII85-encoded images embedded in a level 2 PostScript document.

6 Installing a basic Dienst system

This section⁵ of the Dienst installation instructions describes how to install a minimal Dienst server. This is a server that only indexes a collection of bibliographic files, and does not provide access to the body of documents. All directory names, except when noted otherwise, are relative to the `dienst` directory that you create when you download the Dienst server.

6.1 Considering copyright issues

Before you make your documents available on the Internet, we strongly recommend that you think about the copyright issues. Appendix B describes our approach to copyrights and document availability over the Internet.

6.2 Obtaining publisher IDs for your site

Dienst requires that each document in the entire (distributed) collection have a unique location-independent name, known as a document identifier, or *docid*. A docid consists of a *publisher name* followed by a colon and *document name*. For example, the Cornell docid `CORNELLCS:TR94-1418`, consists of a publisher, `CORNELLCS` (the Cornell Computer Science department), and a document name, `TR94-1418`.

The publisher name uniquely identifies a *set* of documents. What defines a set of documents is up to you. You may wish to specify a single publisher for all of your documents and then use some portion of the document name, such as a *series*, to define subsets within that publisher. Or, you may wish to specify multiple publishers. Your server may provide access to the documents of one or more publishers. To propose and secure a publisher name, or names, for your site, please contact Rebecca Lasher, rlasher@forsythe.stanford.edu. Your proposal should include a short, one word canonical name (e.g., `CORNELLCS`) for the publisher(s) and a longer multi-word user friendly name (e.g. Cornell Computer Science) for each canonical name.

A document name uniquely identifies *each* document within its respective publisher. It is your responsibility to ensure that all of your documents have unique names. The format of the document name is up to you. A document name may have additional structure. For example, at Cornell the document name has three logical parts: a *series* identifier (e.g. `TR`), a *year* (e.g. `94`), and a *serial number* (e.g. `1418`).

6.3 Creating your document database

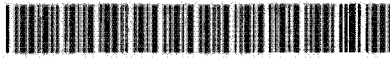
A document database resides on your file system and consists of a directory structure to house your documents in multiple formats. We will describe one format here, the bibliography format. Other formats are described in section 7.

⁵Note: Sections 6, 7, 8, 9, and 10 are copies of the installation instructions that are available on-line at: http://cs-tr.cs.cornell.edu/dienst_install/installation.html. Use the on-line copy for the most up-to-date version.

Exhibit 41

**THIS EXHIBIT HAS BEEN
REDACTED IN ITS ENTIRETY**

Exhibit 42



ADA292221

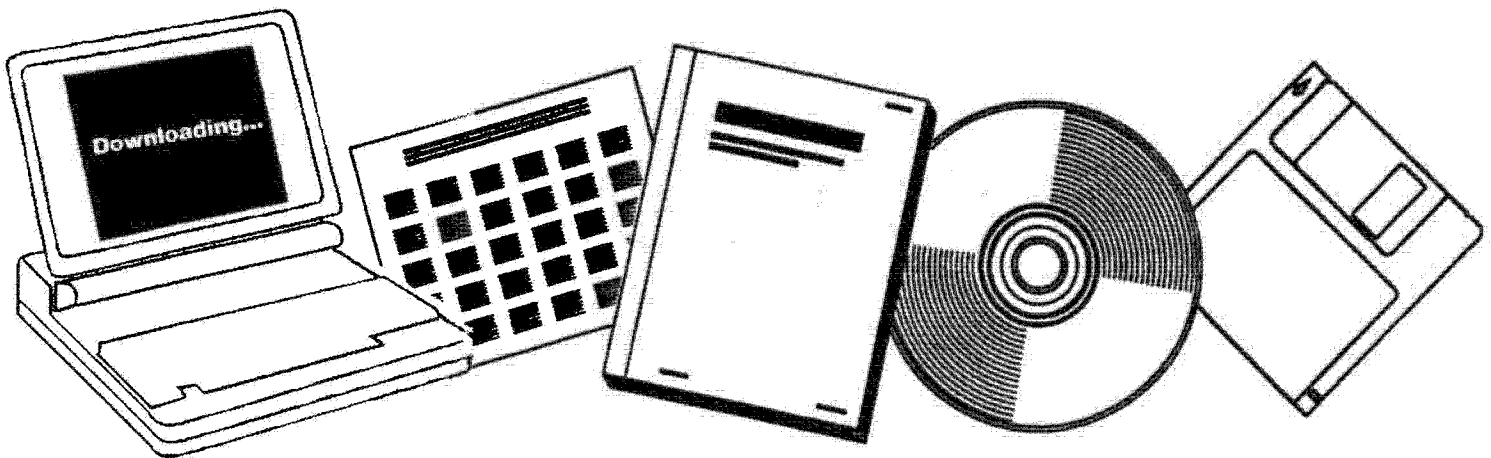
NTIS

One Source. One Search. One Solution.

IMPLEMENTING DISTRIBUTED SERVER GROUPS FOR THE WORLD WIDE WEB

CARNEGIE-MELLON UNIV., PITTSBURGH, PA.
SCHOOL OF COMPUTER SCIENCE

25 JAN 1995

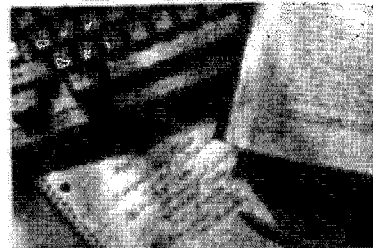


U.S. Department of Commerce
National Technical Information Service

ORCL01818175

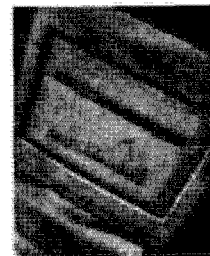
One Source. One Search. One Solution.

NTIS



**Providing Permanent, Easy Access
to U.S. Government Information**

The National Technical Information Service is the Nation's largest repository and disseminator of government-initiated scientific, technical, engineering, and related business information. The NTIS collection includes almost 3 million information products in a variety of formats: electronic download, online access, DVD, CD-ROM, magnetic tape, diskette, multimedia, microfiche and paper.



Search the NTIS Database from 1990 forward

More than 600,000 government research information products have been added to the NTIS collection since 1990. All bibliographic entries for those products are searchable on the NTIS Web site at www.ntis.gov.

Download Publications (1997 - Present)

NTIS provides the full text of many reports received since 1997 as downloadable PDF files. When an agency stops maintaining a report on its Web site, NTIS still offers a downloadable version. There is a fee for each download of most publications.

For more information visit our website:

www.ntis.gov



U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Technical Information Service
Springfield, VA 22161

Computer Science

Implementing Distributed Server Groups for the World Wide Web

Michael Garland, Sebastian Grassia, Robert Monroe, Siddhartha Puri

25 January 1995
CMU-CS-95-114

DTIC
ELECTE
MAR 20 1995
S G D

**Carnegie
Mellon**

19950317 138

DTIC QUALITY INSPECTED 1

DISTRIBUTION STATEMENT A

Approved for public release;
Distribution Unlimited

DISCLAIMER

- ❖ This document has been reproduced from the very best copy that was furnished by the Source Agency. It is being released in order to make available as much information as possible.

Implementing Distributed Server Groups for the World Wide Web

Michael Garland, Sebastian Grassia, Robert Monroe, Siddhartha Puri

25 January 1995
CMU-CS-95-114

School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213-3890

STIC
ELECTE
MAR 20 1995
S G D

Abstract

The World Wide Web (WWW) has recently become a very popular facility for the dissemination of information. As a result of this popularity, it is experiencing rapidly increasing traffic load. Single machine servers cannot keep pace with the ever greater load being placed upon them. To alleviate this problem, we have implemented a distributed Web server group. The server group can effectively balance request load amongst its members (within about 10% of optimal), and client response time is no worse than in the single server case. Client response time was not improved because the measured client traffic consumed all available network throughput. The distributed operation of the server groups is completely transparent to standard Web clients.

This research is sponsored in part by the Wright Laboratory, Aeronautical Systems Center, Air Force Materiel Command, USAF, and the Advanced Research Projects Agency (ARPA) under grant F33615-93-1-1330. The US Government is authorized to reproduce and distribute reprints for Government purposes, notwithstanding any copyright notation thereon. Support also came from the Air Force Materiel Command under contract number F19628-93-C-0171. Views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of Wright Laboratory or the United States Government.

1
2
3

By _____	
Distribution / _____	
Availability Codes	
Dist	Avail and / or Special
A-1	

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

ORCL01818179

Keywords:World Wide Web, distributed computing, hypertext transfer protocol (HTTP), load balancing, server groups, distributed information systems.

1. Introduction

As the exponential growth of the World Wide Web continues with no near-term end in sight, the load placed on popular Web servers is often too great for a single server machine to handle. When the volume of requests to a server becomes prohibitively high the server tends to drop requests and eventually crashes. The service provider needs more computing power to meet the demands of the clients requesting the service. One solution is to simply buy a bigger machine that can handle the large load of service requests. This solution is, however, unacceptable; it provides only temporary relief from server overload. If server usage continues to grow rapidly, even the more powerful server will eventually be overwhelmed. This problem is likely to become more significant over the next few years as the number of commercial service providers and the size of the data objects being exchanged are likely to increase dramatically.

A fairly obvious solution to this problem is to implement a distributed Web server which will spread incoming request load among several machines. There is, however, a significant problem with this approach. The HyperText Transfer Protocol (HTTP/1.0) used by Web clients and servers identifies a resource on the Web using a valid hostname. Consequently, a distributed server needs to present a single hostname to maintain compatibility with existing Web clients. We can alleviate this problem by extending the concept of distributed process groups to the Web server level. A distributed server group exports a single logical name and address to the outside world. From the viewpoint of a client sending an HTTP request, the server group is a single server that will handle the HTTP request in the same way that any other Web server would. Once the server group receives a request it transparently allocates it to an appropriate member of the group for processing.

By supporting distributed Web server groups, we can provide a mechanism for smoothly scaling the server capacity of Web service providers. The use of server groups can reduce the latency of servicing a request provided there is sufficient available network bandwidth to accommodate higher request throughput. Our Web server group implementation can effectively balance request load, and is transparent to Web clients which support the HTTP/1.0 protocol.

2. The World Wide Web

2.1. The HTTP Protocol

The fundamental mechanism underlying the Web is the HTTP protocol. The protocol is stateless, object-oriented, and textual. A complete description of the current protocol can be found on-line at [CERN]. For our purposes, we are only interested in the basic structure of the protocol.

HTTP is an RPC-like protocol; clients make requests of servers and receive responses. The general format of a request is:

method	URL	protocol version	CrLf	GET /doc.html HTTP/1.0 <CrLf> <CrLf> <end>
optional header information			CrLf	
Body				

Figure 1: HTTP request format.

Figure 2: Example HTTP request.

This figure needs some explanation. A *URL* is simply a name for an object; typically, the hierarchical naming scheme of the Unix file system is used. The *method* given in the request is a method to be invoked on the object named by the given URL. The *CrLf* tag represent the pair of characters carriage return and line feed. The *optional header information* is a series of MIME headers which can be used to specify a variety of things including client name, user name,

encoding type, date, authorization information, etc. The *body* of the request is interpreted by the method which is invoked; it is in essence the argument list for the method.

Once the server receives a request from a client, it processes the request and returns a response. A reply from the server has the following structure:

protocol version	status code	reason	CrLf	HTTP/1.0 200 OK <CrLf>
optional header information			CrLf	Server: NCSA/1.3 Content-type: text/html <CrLf>
Body				HTML document <end>

Figure 3: HTTP response format.

Figure 4: Example HTTP response.

The *status code* indicates the result of the request; the *reason* field provides a human-readable textual explanation of the status code. The rest of the fields of the response fulfill similar roles to their counterparts in the request.

In current patterns of Web use, the GET method is by far the most frequently used. In fact, Web sessions typically consist exclusively of GETs. This reflects the fact that the Web is currently used primarily as a means of fetching files, albeit in many formats, including hypertext.

2.2. Web Servers and Clients

Servers and clients use the HTTP protocol described above to exchange information. The general model of the Web currently in use is this: servers export a hierarchical file space to clients. This file space is populated by "documents" of various types. The primary function of Web clients is to acquire documents from this file space in response to user actions. The client and server must also negotiate a document format which is acceptable to them both. Web servers and clients communicate with each other over TCP/IP streams. The general model of an HTTP transaction is:

- The client opens a TCP/IP connection to the server,
- The client sends its request over this stream,
- The server sends its response back,
- And the connection is closed.

In the case of the most widely used Web server (NCSA's `httpd`), the server forks a new process for each connection. That process receives the request from the client, processes it, sends back a response, and exits.

3. A Distributed Web Server Group

Since its inception, the Web has experienced very rapid growth in usage and amount of data being provided. The growth of the user community has generated increasing demand for data, and the raw amount of data has also increased due to the proliferation of large multimedia resources. The result has been continually increasing stress on Web servers.

A natural solution is to group several server machines into a single server group. The server group would provide a single logical service but would physically share incoming requests amongst its member machines. The need for this is indeed real; NCSA has implemented a similar scheme for their server after a single server machine was no longer able to cope with the heavy load [KBM94].

In order for a Web server group to be useful it must operate transparently within the existing Web. In other words, it

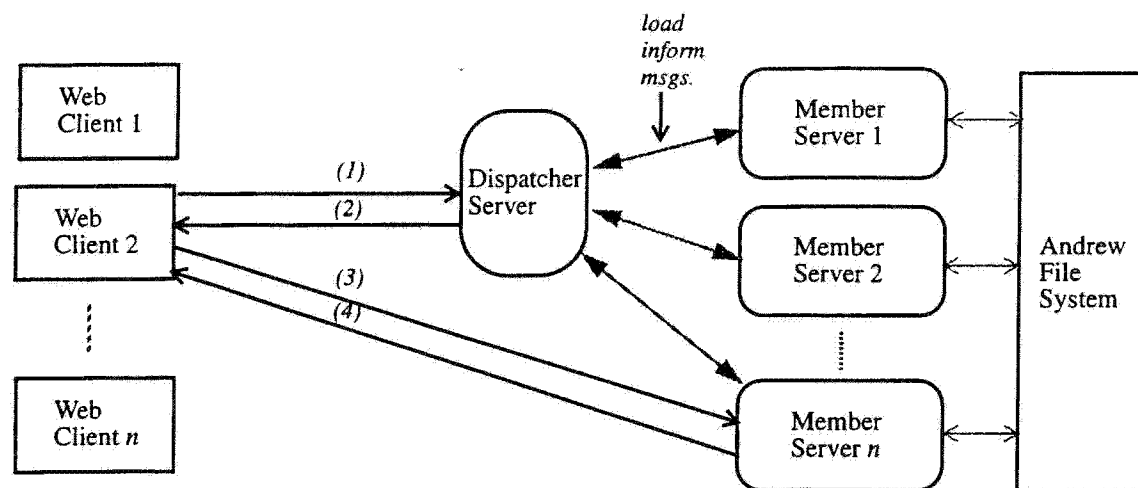


Figure 5: Architecture and Protocol for the Distributed Web Server Group. (1) Client sends an HTTP request. (2) Dispatcher calculates least-loaded server and sends forwarding address. (3) Client resends HTTP request to forwarding address (member server). (4) Member server satisfies request.

must be implemented within the framework of the HTTP protocol and it must be completely compatible with existing Web clients. The server group implementation is also constrained by the current URL specification. A server group must be identified by a unique Internet address. In addition to all this, we also want the server group to perform well: it must balance request load effectively and introduce as little new overhead as possible.

3.1. System Architecture

We have chosen the following general architecture for our Web server groups:

- There is a special server (the *dispatcher*) which coordinates and controls the server group. Requests intended for the server group are received by the dispatcher.
- Attached to the dispatcher is a collection of *member* servers. These servers are the actual repository of data and fulfill requests made of the group.

The dispatcher is the single entity representing the group to the outside world. Incoming requests are received by the dispatcher, and they are redirected to one of the member servers for actual processing. It is also the task of the dispatcher to balance the load placed upon the various members under its control. In essence, it tries to always dispatch incoming requests to the least loaded member. Each member exports an identical copy of some data repository. In our case, this data is shared via a common AFS directory. Figure 5 diagrams both the system architecture and communication protocol.

We have implemented our new servers as extensions to the NCSA Web server (`httpd`). This server has several attractive properties: it is well supported, it is in wide use, and it is quite portable. One major consequence of this choice has affected our design; since the `httpd` server forks a new process for every connection, a process handling an incoming request cannot modify the global server state.

We have decided that the dispatcher should not fork on connection like the standard `httpd` server; it only forks when sending messages to its members. This lowers the latency of request dispatching and allows the global server state to be modified during request processing. Throughout the design process we felt that the dispatcher was likely to be a bottleneck, since it is a single point through which all requests flow. Processing each request serially before accepting the next connection introduced an even greater danger of creating a bottleneck. Therefore we have tried to keep the amount of processing performed by the dispatcher to a minimum. Fortunately, in our experience the load on the dispatcher never reached a critical point. It remained only marginally loaded.

3.2. Interfacing to Web Clients

In order to make the server group work, there needs to be some mechanism for the dispatcher to redirect a client's request to another server. Fortunately, such a mechanism already exists and is supported by all major Web clients (including Mosaic, Lynx, and W3).

Recall that every response from a server to a client includes a status code. The status code of particular interest to us is: Found (302). It instructs a client that the requested object has been found on another server, whose address is included in the body of the response. The client is expected to re-request that object from the given alternative server. This process of redirection is completely invisible to the user.

3.3. Protocol Extensions

To support server groups, we needed some means for communication between the dispatcher and its members. One possibility would be to build a dedicated protocol for this purpose, possibly using UDP instead of TCP. However, we decided to implement this communication through extensions to the HTTP protocol. As a result, the intragroup communication fits smoothly within the overall structure of the server. This has a number of advantages; the chief advantage being that our code should be easily portable to future server versions as well as other servers.

The extensions we have made to the HTTP protocol are fairly minor. Most of these changes involve simply adding new methods, and they only affect the dialog between the dispatcher and its members. These extensions are completely invisible to clients. The first set of extensions involves adding methods for administering server groups. In these cases, the dispatcher is acting as a server and a member is communicating with it in the role of a client. The new methods are as follows:

Method: GROUP-ADD

Body: URL of a member server

This method adds the specified member server to the server group administered by the dispatcher.

Method: GROUP-EXIT

Body: URL of a member server

This method removes the specified member server from the dispatcher's server group.

Method: GROUP-QUERY

Body: None

Returns status information about the server group and its members. Currently this is meant only for debugging purposes. A real system might choose to ignore this request.

The next extension actually diverges from the HTTP specification. It is implemented as a *message* rather than as a request. A response is neither required nor expected. The format of the message is:

Method: LOAD-INFORM

Body: Load value and time stamp

This message communicates load information from a member to the dispatcher. This information is then used by the dispatcher in its load balancing decisions.

Our final extension is also a message rather than a request.

Method: LOAD-QUERY

Body: None

This message is used by the dispatcher to query load information from a member. The member is expected to respond later with a LOAD-INFORM message.

This last requirement may seem slightly strange; the member is required to respond to a LOAD-QUERY message with another message rather than a direct response. The reason for this is related to the structure of the dispatcher. Since the dispatcher forks when sending messages, any response would be unable to affect the global state. To get around this difficulty, we require the member to send a LOAD-INFORM back to the parent process.

3.4. Load Balancing

The task of the dispatcher is to distribute incoming requests to members of the server group. However, simply doing this without making informed decisions about *where* to send requests would not be very desirable. We want the dispatcher to distribute requests in such a way as to balance the load placed on the various member servers. This requires two things: the dispatcher must have some knowledge of the load on members, and there must be some scheduling policy used to select which member to dispatch the next request to.

The first issue that needs to be addressed is how to measure the load on a member server. At first glance, you might suspect that the dispatcher should have enough information to compute the load on a member. After all, every request that arrives at a member was directed there by the dispatcher. In reality, this is not true. When examining actual HTML documents it becomes evident that many links are either relative to the current server or point to a remote server. This has the following consequence: when first contacting a server group, a client will be dispatched to a particular member. Subsequently, all relative links will be resolved by going *directly* to that member. This property actually has the desirable effect of helping to balance the load [CHE91]. In addition, since each client connection is for a single request, there is no way of predicting how long a particular client will continue making requests of the server. The dispatcher cannot know the load on a member, so it must be informed of the member's load (using the LOAD-INFORM message described above).

A good load measure for the members should account for two sources of load: the traffic of requests and the general load on the machine. Obviously, request traffic is an important source of load. Including the load on the machine as well is necessary because there might easily be other active processes on the same machine. In particular, there might be multiple members per machine. The formula we use to compute the load on a member is $Load = 10(r + m)$ where r is the number of requests per second being received by the member and m is the machine load as measured by the number of processes in the run queue. The scaling factor of 10 is used to allow the load to be represented as an integer, for faster processing. Note that with our servers, there is a direct correlation between requests being processed and number of processes. This load information is transmitted periodically to the dispatcher (every 15 sec. in our experimental configuration).

The dispatcher uses a simple scheduling policy. It keeps all the members in a doubly-linked priority queue, sorted by increasing load. Thus, dispatching a request simply involves looking at the head of the queue and returning that member address to the requesting client. The dispatcher also attempts to estimate the change of member load over time by adding a constant factor to the load every time a request is dispatched to that member. If the dispatcher does not receive a LOAD-INFORM query from a member after some long amount of time, it will send a LOAD-QUERY message to it. If this fails for some reason, the member is assumed to have failed and is removed from the server pool.

4. Performance Analysis

4.1. Experimental Overview:

The most appropriate test for our distributed Web server would involve observing its performance in a heavily accessed location over a period of months. However, practical considerations obviously preclude this type of testing from the scope of this project. Instead, we have attempted to simulate heavy (localized) load on the servers and observe how they perform under a variety of file access patterns. We have attempted to use file access patterns which we feel are fairly representative of common Web usage patterns. Our testing has focused on the latency for satisfying

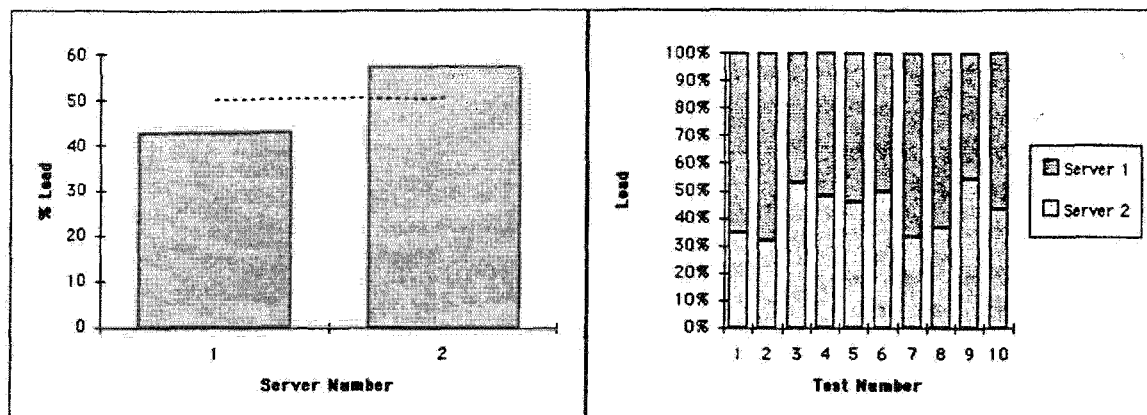


Figure 6: Total Load Balance (2 servers)

Figure 7: Instantaneous Load Balance

a request (from the user's perspective), and on the balancing of load amongst the group members. These metrics reflect the perception of a user browsing the Web, and server hardware utilization, respectively.

4.2. Experimental Procedure:

We developed a test suite of 125 tests to observe how the server groups performed as various aspects of the system were stressed. As a control test, we performed each of the test series on a vanilla `httpd` server. To measure server performance, the member servers were instrumented to periodically dump their load information into log files.

To measure request latency as seen by clients, we constructed a programmable HTTP client (using Tcl) that measured the time required to satisfy an HTTP request. The phases timed by the client were: connection to the dispatcher, redirecting to the member server, and finally receiving the requested data. All timing measurements were made in C code to avoid Tcl overhead, and the data read by the client was not interpreted in any way. Many machines were required to test system performance; the final test suite was performed in Carnegie Mellon's computing labs early in the morning over Thanksgiving break, 1994. The campus network was very lightly loaded. We used 10 Sun Sparc-5 machines as clients and 5 SGI Indigo2 machines as servers. The entire test suite took about five hours to complete. External load on the network remained consistently negligible, but present, throughout the duration of the test.

4.3. Test Cases:

The three parameters which were varied during our tests were: the number of member servers, the number of concurrent clients, and the file access pattern. Specifically, we looked at 1 vanilla `httpd` server, 1, 2, and 4 member servers, 1, 5, and 10 clients, and ten different file access patterns. The file access patterns were designed to simulate common Web access patterns. All of the file access patterns were some combination of small (4KB), medium (100KB), and large (2MB) files. These file sizes were designed to represent a small HTML home page, a medium JPEG image, and a short MPEG video clip, respectively. We ran test cases for each possible combination of server count, client count, and file access pattern.

4.4. Experimental Results

Server Performance Observations: The long term load balancing performance of the servers is displayed in Figures 6 and 8. As you can see, the dispatcher was able to maintain reasonable load balancing among the member servers. The dashed line on each graph indicates the optimal case. As the graphs show, the actual load observed was within 7% of optimal with 2 servers and within 9% of optimal with 4 servers. We consider this performance to be quite acceptable. Figure 7 shows a more fine grained load balancing analysis over ten consecutive test cases with 2

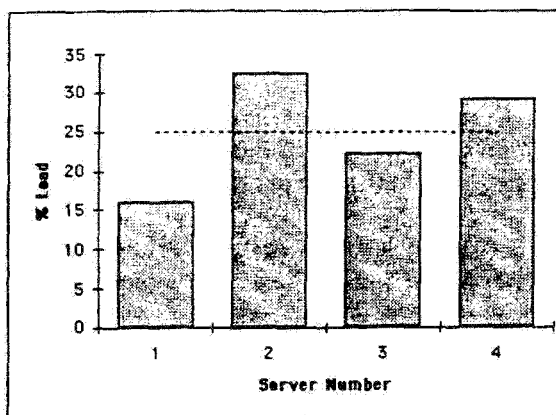


Figure 8: Total Load Balance (4 servers)

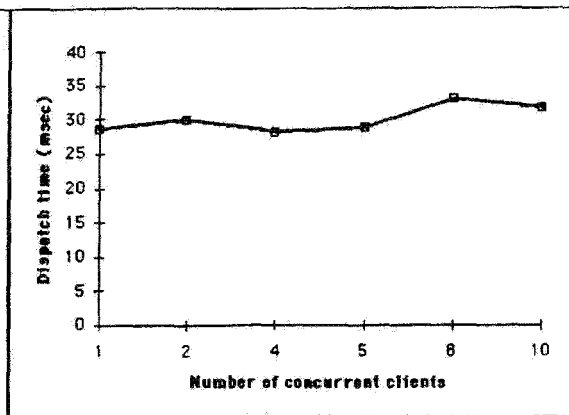


Figure 9: Dispatch time with increasing client load.

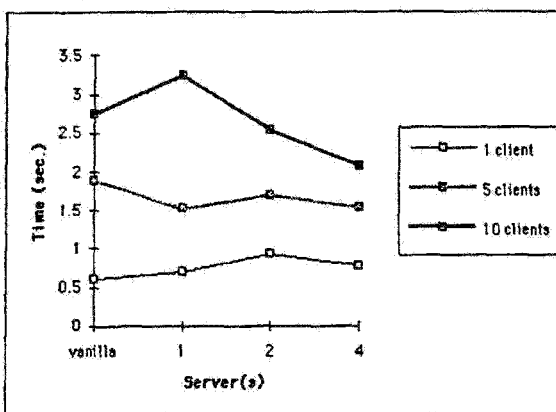


Figure 10: Time to access 10 4K files.

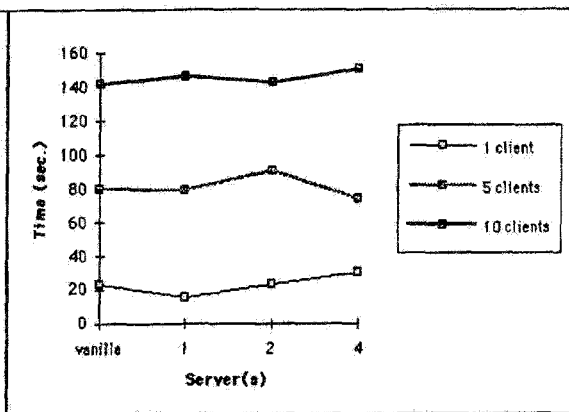


Figure 11: Time to access 5 2Mb files.

servers. As you can see, the instantaneous load balance varies somewhat. This could be improved by reducing the granularity of LOAD-INFORM messages from the members, but this would probably lower total throughput. It is practically impossible to ensure optimal load balancing. Once a client has a member's address, it can contact that member any number of times; these accesses cannot possibly be accurately predicted by the servers.

User Performance Perception: Unfortunately, the performance witnessed by the clients was not as promising as the load balancing results. While conducting the tests it became rapidly apparent that the bandwidth of the Ethernet was insufficient to keep up with even a small number of clients and servers, let alone a barrage from four servers and ten clients. Throughout the tests, it was not uncommon for the number of Ethernet collisions in the server pool to exceed the number of successful outgoing packets. As a result, the client latency figures are fairly erratic, and show only a slight improvement as servers are added. Figures 10 and 11 show request latency for very small and very large file access patterns. The curves are somewhat erratic. Adding servers to the group did not generally improve request latency, but it was also generally no worse than that of the stand-alone vanilla server. It would certainly seem that in our experimental setup, the network, and not the server machines, was the chief limiting factor on performance.

Dispatcher Server Performance: One of the promising results of our experiments was the performance of the dispatcher. We had anticipated that the dispatcher would be the primary bottleneck in our experiments; however, it turned out to be remarkably fast and scalable. We attempted to determine where the dispatcher would break by having 1 to 10 clients send a barrage of requests to the dispatcher; the forwarding responses were merely discarded. With the same 10 client and 4 server configuration that rapidly saturated the Ethernet when data was actually transferred, we were unable to force the dispatcher to refuse any connections or dramatically increase the time required to process a request. As Figure 8 shows, the time required to process a request stays fairly constant at around 30 milliseconds, even as the number of clients is increased from 1 to 10. This was a surprising result. With the 10 client and 4 server machines we had at our disposal, we were unable to significantly degrade the dispatcher's performance. Furthermore, the 30 msec. overhead induced by the redirection is small enough to be imperceptible to a human user. Obviously, the

30 msec. delay is very sensitive to network distance travelled, but the results paint a very encouraging picture of the general cost of dispatching.

5. Related Work

Distributed process groups have been explored in the ISIS system [BIR91] and the V system [CHE85]. These two systems implement slightly different forms of process groups, neither of which are sufficient to completely solve our problem. However, they do establish the basic concepts of distributed process groups, and hence distributed server groups.

Load balancing methods can be broken down into two main categories: those based on communication and those based on dispatching patterns [HSU86]. In the first category, a group of equal servers dispatches jobs to its members, trying to optimize (or at least improve) some performance criteria. This requires that each server be kept informed of the load on every other server. The second approach views the problem as one of routing, where a central dispatcher routes all jobs to servers based on a routing policy which may involve optimization. The critical difference for our purposes is that in this scheme the servers whose loads we are trying to balance need to communicate only with the dispatcher. Schemes for dynamically scheduling requests to balance load have proven successful in several application areas similar to our own [ALO87, WIK91, HSU86, PAR92].

The NCSA has also implemented a distributed Web server mechanism [NCSA]. Their server, which is probably the most heavily loaded server on the Web, was beginning to experience actual system failures due to extreme load. They report that the system was incapable of handling more than 20 requests per second without crashing. To solve this problem, they extended the BIND name resolution server at their site. It now allows them to specify multiple physical machines for the same logical name. When HTTP connections are requested on the logical name, the BIND server resolves that name to one of the physical machines. Machines are scheduled in a purely round-robin fashion. They report that load was not very well balanced (1 of their 4 machines consistently got 50% of the load), but it did solve their problem of server failure.

6. Future Directions

There are several ways in which our work could be extended in the future. To support active use in the Web at large, there are a few features which would make our servers more complete. As with most experimental systems, our code is not in a state acceptable for widespread use. There are also areas in which our servers could be extended or altered to improve functionality and performance.

As they are currently implemented, our servers implement the basic server group functionality. However, there are several extensions that could be made. The GROUP-ADD and GROUP-EXIT requests could be extended to allow multiple servers to be added/removed in a single request. We could also extend the dispatcher to support anonymous members. It is conceivable that some servers might not want to advertise their location to the rest of the world. In this case, the dispatcher could serve as an intermediary between the client and the anonymous server. A request for anonymity would be made in the GROUP-ADD request. Currently, every member exports identical data. It would be nice to allow specific members to export data objects unique to themselves. This would complicate the dispatcher in that it would have to actually interpret the URLs handed to it, look them up in a table, and determine whether they had a special location or could be dispatched to anyone. With the current, non-forking, architecture of the dispatcher, one would need to carefully consider whether this name resolution would add unacceptable overhead.

This point highlights what is probably the most important improvement which should be made to our servers, and indeed to all Web servers. They should be multithreaded. Servers currently pay a high price for accepting a connection, a price particularly noticeable during small requests. In addition, the server process interacting with the client cannot modify the global server state. A multithreaded server should perform significantly better, and obviate the contortions one must go through to maintain server global state.

An important area of inquiry which was beyond the scope of our current project, but which could be very informative, is the long term measurement of server group performance under actual use. Although we have tried to examine "reasonable" server usage, our experiments have used inherently artificial access patterns. It is nontrivial to predict just how our measured performance would translate into actual performance. With our fairly limited time and performance data, we were unable to seriously explore variations in the actual dispatching policy. Such inquiry might lead to significant performance improvements.

Our tests show that the constraints of network bandwidth overshadow the constraints of CPU load for our distributed server. This suggests that we could incorporate more intelligent request processing into our servers, and the added computation would not impair performance. We could also allow each server to accept a higher CPU load by providing more intensive services such as numerical computation or database lookup, instead of simply sending files. In this case we would expect our distributed servers to provide a significant performance enhancement.

The most obvious way to avoid a network bottleneck is to distribute the servers geographically so that they reside on separate networks. The dispatcher would then take network load and proximity into account when deciding which server to transfer a client to. This gives rise to the paradigm of "transparent mirrors," which could be seen as extending the current practice of geographically distributed mirror sites by providing a transparent centralized directory service. The most difficult problem that this approach raises is determining proximity. While some way of calculating geographical distance may be a good heuristic, what we would really like is a distance metric based on network topology. Finding a good way for the dispatcher to dynamically keep track of the topology of the internet and the probable available bandwidth at each link is wide open for further study.

7. Conclusion

The increasing load placed upon Web servers is clearly a serious problem. The NCSA has already become so overloaded that they were forced to deploy a distributed server mechanism. While their method works, it is not particularly clean, and it does not balance request load very well. Server groups provide a much nicer abstraction for distributed Web servers, and they support a wider range of possible uses. With our current architecture, server groups could also be deployed without any changes to system software as required by the NCSA approach.

Our implementation demonstrates that server groups are both feasible and effective. Server group functionality can be added to existing HTTP servers without modifying the overall structure of the servers. Our servers have also proven capable of distributing request load fairly amongst group members. For all this, the server group does not incur any significant cost in client response time.

8. Bibliography

Process groups:

- [BIR91] Birman, Kenneth; The Process Group Approach to Reliable Distributed Computing; *Communications of the ACM*, December 1993, pp. 36-53.
- [CHE85] Cheriton, David R., and Zwaenepoel, Willy; Distributed Process Groups in the V Kernel; *ACM Transactions on Computer Systems*, May 1985, pp 77-107.

Load Balancing:

- [ALO87] Alonso, Rafael; An Experimental Evaluation of Load Balancing Strategies; Technical Report CS-TR112-87, Department of Computer Science, Princeton University.
- [CHE91] Cheng, William C., Muntz, Richard R.; Optimal routing for closed queueing networks. *Performance*

Evaluation, vol. 13, no.1; Sept. 1991 pp. 3-15.

- [HSU86] Hsu, Chi-Yin Huang, Liu, Jane W.S.; Dynamic Load Balancing Algorithms in Homogenous Distributed Systems; Technical Report UIUCDCS-R-86-1261, Department of Computer Science, U of Illinois, Urbana-Champaign.
- [LIN91] Lin, H.-C., Raghavendra, C.S.; A dynamic load balancing policy with a central job dispatcher. Proceedings of 11th Intl. Conf. on Distributed Computing Systems; May 1991; pp. 264-271.
- [PAR92] Parris, Colin, Zhang, Hui, Ferrari, Domenico; A Mechanism for Dynamic Re-routing of Real-time Channels; Technical TR-92-053, Computer Science Division, UC Berkeley.
- [WIK91] Wikstrom, Milton C. et al; A Meta-Balancer for Dynamic Load Balancers; Technical Report TR#91-04, Department of Computer Science, Iowa State University.

HTTP and Distributed Web Servers:

- [CERN] HTTP specification -- working document specifying the hypertext transfer protocol (HTTP) 1.0, available on the WWW at the URL "<http://info.cern.ch/>".
- [KBM94] Katz, Butler, McGrath: "A Scalable HTTP Server: The NCSA Prototype" - working document from NCSA. Available on the WWW at the URL "<http://www.elsevier.nl/cgi-bin/query/WWW94/FinalProgramme.html?katz>".

ALL SALES ARE FINAL

NTIS strives to provide quality products, reliable service, and fast delivery. Please contact us for a replacement within 30 days if the item you receive is defective or if we have made an error in filling your order.

▲ **E-mail: customerservice@ntis.gov**

▲ **Phone: 1-888-584-8332 or (703)605-6050**

Reproduced by NTIS

National Technical Information Service
Springfield, VA 22161

***This report was printed specifically for your order
from nearly 3 million titles available in our collection.***

For economy and efficiency, NTIS does not maintain stock of its vast collection of technical reports. Rather, most documents are custom reproduced for each order. Documents that are not in electronic format are reproduced from master archival copies and are the best possible reproductions available.

If you have questions concerning this document or any order you have placed with NTIS, please call our Customer Service Department at 1-888-584-8332 or (703) 605-6050.

About NTIS

NTIS collects scientific, technical, engineering, and related business information – then organizes, maintains, and disseminates that information in a variety of formats – including electronic download, online access, DVD, CD-ROM, magnetic tape, diskette, multimedia, microfiche and paper.

The NTIS collection of nearly 3 million titles includes reports describing research conducted or sponsored by federal agencies and their contractors; statistical and business information; U.S. military publications; multimedia training products; computer software and electronic databases developed by federal agencies; and technical reports prepared by research organizations worldwide.

For more information about NTIS, visit our Web site at <http://www.ntis.gov>.

NTIS

**Ensuring Permanent, Easy Access to
U.S. Government Information Assets**



U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Technical Information Service
Springfield, VA 22161 (703) 605-6000