A Middleware System between WEB and Database Servers

Mohammad H. Abu-Arqoub, Ihab S. Serhed, Waheeb A. Abu-Dawwas, and Rashid M. Al-Azzeh

***Abstract*—**This paper aims at improving web server performance by establishing a middleware layer between web and database servers, which minimizes the overload on the database server. A middleware system has been developed as a service mainly to improve the performance. This system manages connection accesses in a way that would result in reducing the overload on the database server. In addition to the connection management, this system acts as an object-oriented model for best utilization of operating system resources. A web developer can use this Service Broker to improve web server performance.

***Keywords*—**Database server, Improve performance, Middleware, Web server.

I. Introduction

CURRENT web servers use various Application Programming Interfaces (API) sets to access backend services. As a result, they do not support overload control, service differentiation, and caching of contents generated by backend servers [1].

Additionally, as a request rate increases beyond the server capacity, the server response-time and connection error rate will increase, and therefore, web servers today tend to offer poor performance under overload [2] [4] [10].

A typical web server environment generally consists of two parts: front-end web server and backend servers. Backend servers are used for dynamic pages, not for static ones [11].

A static page contains embedded objects and has no access to a database server. As a result, all clients receive the same page with the same content.

Current dynamic web applications are formulated as a Common Gateway Interface (CGI) that runs in a separate process. Alternatively, they can be formed as scripting language codes, like JSP and PHP that usually runs in web server processes. These dynamic applications access backend servers through specific APIs such as sockets, Open DataBase Connectivity (ODBC) or modules like Common (Component) Object Model (COM). These APIs reside in the application process' space and share no information with other processes [1].

Huamin and Prasant paper [1] provides suggestions regarding the capability of using one connection for each dynamic application in order to access backend servers such as database, mail and file system. This system becomes overloaded because it uses one connection for each dynamic application, and hence, there is no connections' management. In addition the response time is decreased at the expense of data credibility, and request will be dropped when the number of requests exceeded a specific threshold.

.NET framework develops a connection management and a connection pooling. This framework showed that by utilizing a specified set of connections and pooling them between multiple sessions or clients, developers will be able to increase the scalability and extensibility of their applications [15].

.NET framework uses the pool technique for connection management, but Service Broker uses the pool technique for request management (thread management). The number of connections is limited for each application and increased according to the load with reusability, so while one application is running, all of its limited number of connections, are going to be running too.

II. The Problem's Definition

The major problems can be addressed are managing accesses to the database backend server. Some strategies and organizations are implemented for a web server in a way that would lead to improving the performance.

The Application Programming Interfaces currently used in web applications, reside in the application process' space and share no information with other processes. Hence, the drawbacks of this paradigm are [1]:

- Backend servers become overloaded due to the entire request handling process.
- Access is isolated and not globally optimized.
- The overhead induced by contexts switching.

Therefore, for each dynamic request from clients, the web server does the following:

- Receives the request from the client.
- Establishes a separate connection to the database server that shares nothing with other connections.
- Sends queries to the database server and receives a response.
- Closes the connection and returns the result back to the client.

Nevertheless, connections reside in the web server even if it closed, until the garbage collector deletes them from the memory.

In our solution, for each request to the database, the web server will:

- Receives the request from the client.
- Forwards this request to our Service Broker system, and then the Server Broker will assign it to one of the running threads that follow a parent class to let all running threads be shared for a specific application.

III. Typical Web Server Paradigm

Web server environment typically consists of two parts: front-end and backend. Fig. 1 shows these two parts. In front-end, there is a set of dynamic applications that usually run at known ports. These dynamic applications are used as connectors between clients and backend servers. Front-end waits for requests from clients at specific ports. Once a request arrives, it uses a set of APIs to access backend servers such as database or mail server.

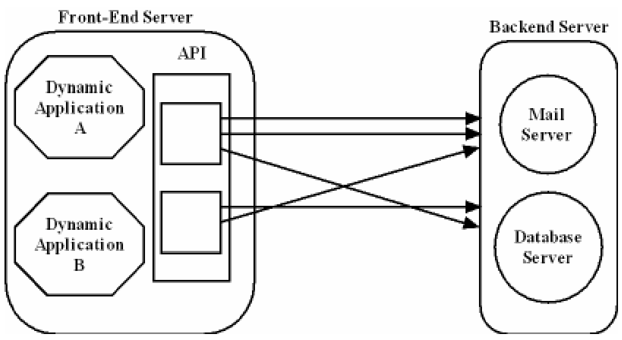

Fig. 1 Typical web server paradigm

The structure in Fig. 2 illustrates the stages of the request process in a typical web server; as shown in this Figure, requests arrive at the front-end web server. Once the request arrives, it will be redirected to the appropriate dynamic application that assigns a thread to handle it.

A Thread is specialized in checking the validity of the requests, besides it uses a database API to access the database server. However, if the request is for a mail server then the thread in dynamic application will use the mail API instead.

The overload on the database server is obviously clear. Moreover, each connection to the database server is completely separated from other connections; the worst case is to have each request, requiring information from the database server, being created another connection. This means that for each request, there is a connection establishment. After serving the request, the connection will be released. In addition, it resides in a system memory for a long time because the operating system does not monitor each connection release. The operating system gathers a collection of released connections and frees them when the system lacks memory.

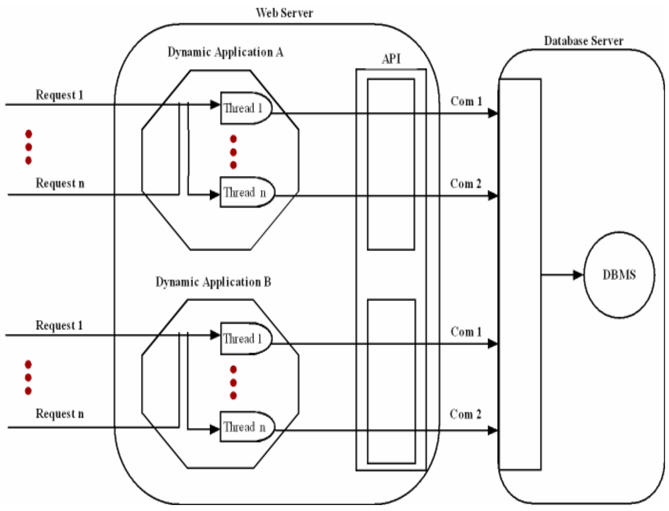

Fig. 2 Stages of request process in typical web server

IV. Web Server Paradigm Using Service Broker System

Service Broker uses a limited number of open connections to the database server, to be used without repeatedly establishing connections, and without shutting down. Fig. 3 illustrates the architecture of a web server using Service Broker. In this Figure, the request arrives to HTTP front-end server. After that, the request is examined and forwarded to the appropriate dynamic application. The dynamic application, in turn, would forward this request to the Service Broker system. Instead of using the current set of APIs, the Service Broker system receives the request on a specific port by the Service Broker listener and makes thread to this request. This thread passes through many stages from authentication to analyzing, and finally to the active application that follows the Service Broker connector.

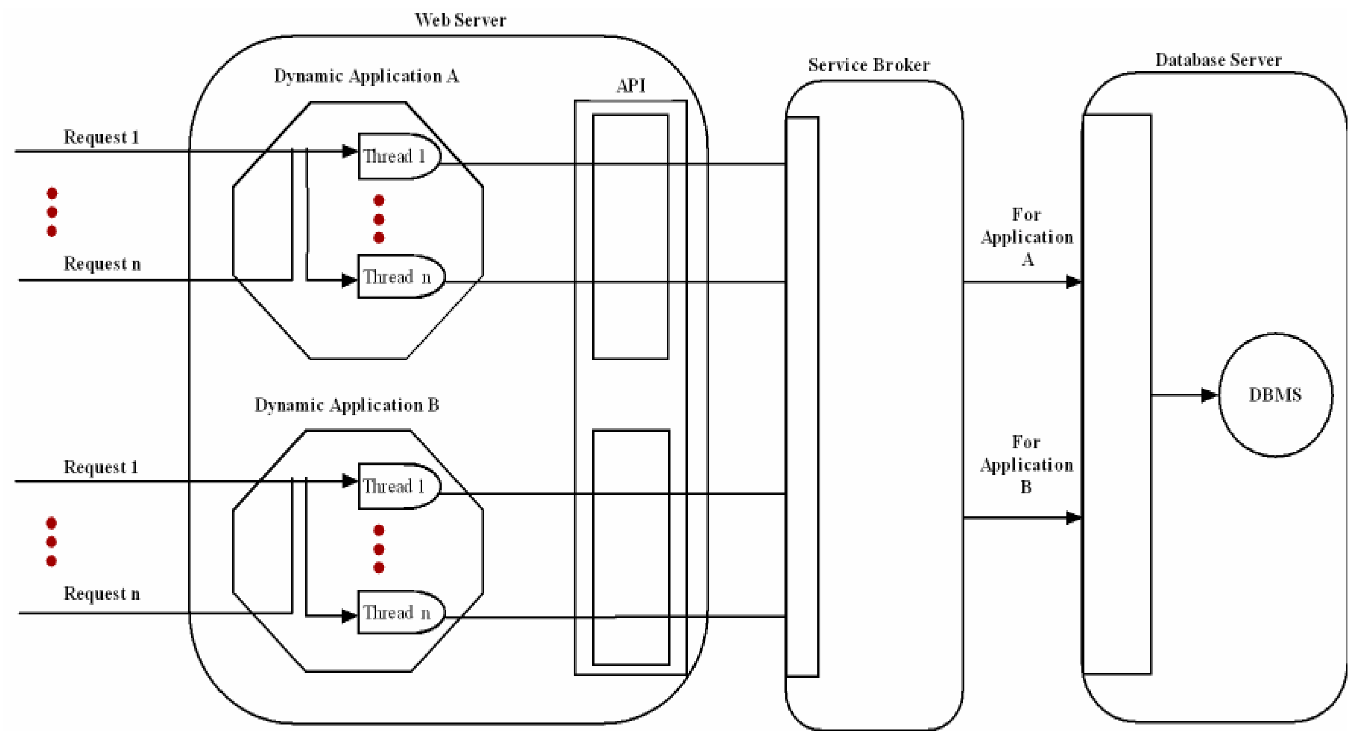

Fig. 3 System Architecture using Service Broker

V. The Proposed Service Broker System

Today's users still expect real-time software that can process data at very high speed. Users have grown by the time and expect to receive responds in seconds regardless of how much data is requested and how much data is handled by the server [4]. The simplest way to improve a website's performance is by scaling up the hardware. However, scaling the hardware does not work in all cases, and costs money as well. Our Service Broker system improves the performance without extra costs for the hardware by providing some recommendations that were shown to be helpful in improving the performance.

Service Broker system is used as a middleware or as an intermediate process between a front-end web server and a backend server instead of API that have to access a database server. It receives messages from a front-end web server, and then it sorts and rewrites these messages and produces a query. After receiving replies from the backend server, Service Broker sends the results to the front-end and makes a local copy to serve similar requests if possible. Instead of making a separate connection for each request, a limited number of connections will be used that can be established in advance to reduce the overhead.

The Service Broker system is implemented as a real time program using Java and results in a Service Broker package as an open source that can be uses and modifies.

Fig. 4 illustrates the Service Broker architecture as an object oriented model. The system consists of the following components:

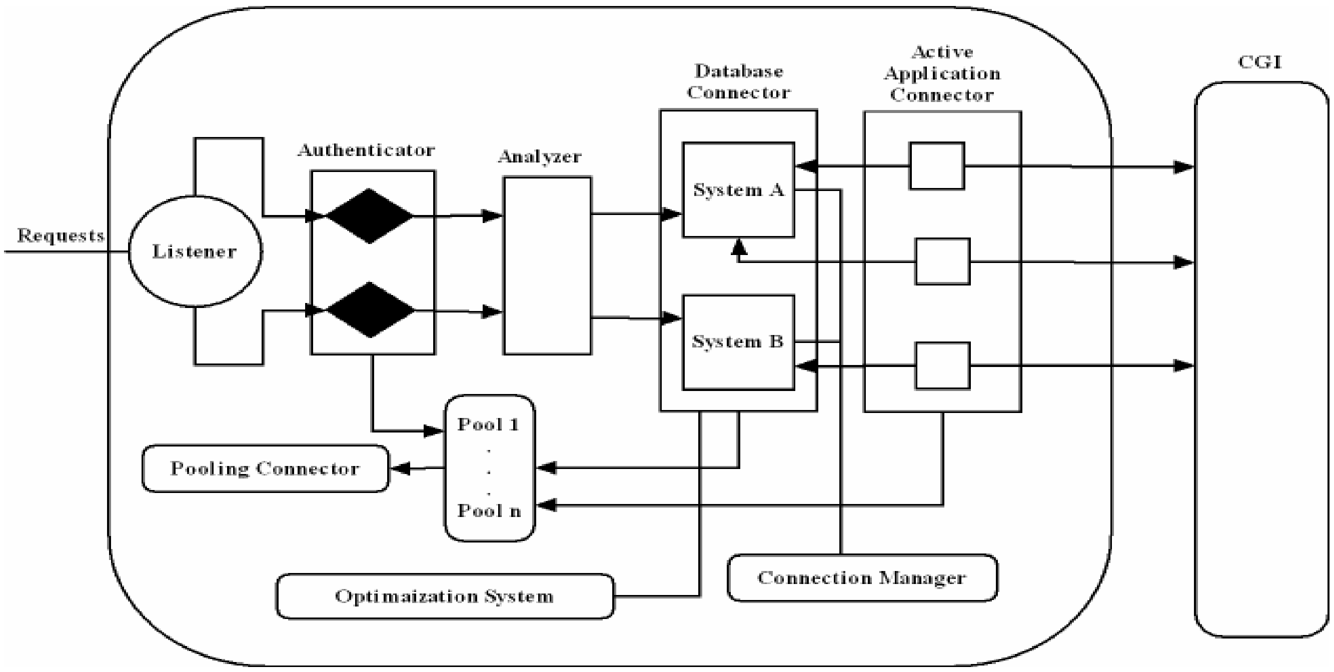

Fig. 4: Service Broker Architecture

A. *Database Connection Management*

Each web application has one object from the ServiceBrokerDBConnector class that contains one object from the ServiceBrokerOpenedConns class.

ServiceBrokerOpenedConns contains five connections to the database, so for each application system, there are only five connections to the database that are always open. These connections are used whenever one of the ServiceBrokerActive applications needs to access the database.

These five connections are used to insert, update, delete, select, etc. The following list describes each connection's task:

- The first connection is used for a workspace operation:
  This connection is used for inserting, updating and deleting instructions. There is no real commit at this connection because it is used as a testing connection. The commit is only applied to the workspace, which is an image from the early database. This connection's job is to prevent the contradiction between the several requests.
- The second connection is used for query instruction:
  This connection is only used with SELECT statements; there is no need to commit in this connection, and this is why it can be easily shared by all requests.
- Three more connections are used in the commit operations:

These three connections are used because every commit instruction holds many SQL statements that need to be executed at a live database. These SQL statements were previously tested in the workspace connection not at a live database. Once this connection sends a commit packet, Service Broker occupies one of these three connections. Each one of these connections has a synchronized access, which implies that only one thread can access this connection at once. A synchronized access has been used because the transaction is atomic, so each bundle of transactions which follow a commit for a specific web server's connection must be committed as one unit; if not possible, and then none of them would be committed.

Fig. 5 illustrates a database connection management for two applications. In this Figure, there are five connections for each application. Each five connections are completely separated from other applications. Each workspace connection from each application has access to the same workspace database, which indicates that the use of workspace prevents contradiction between all applications.

In addition there are pools where each pool contains a set of threads. Fig. 7 illustrates both pools and pool collector to satisfy the factor of reusability of resources.

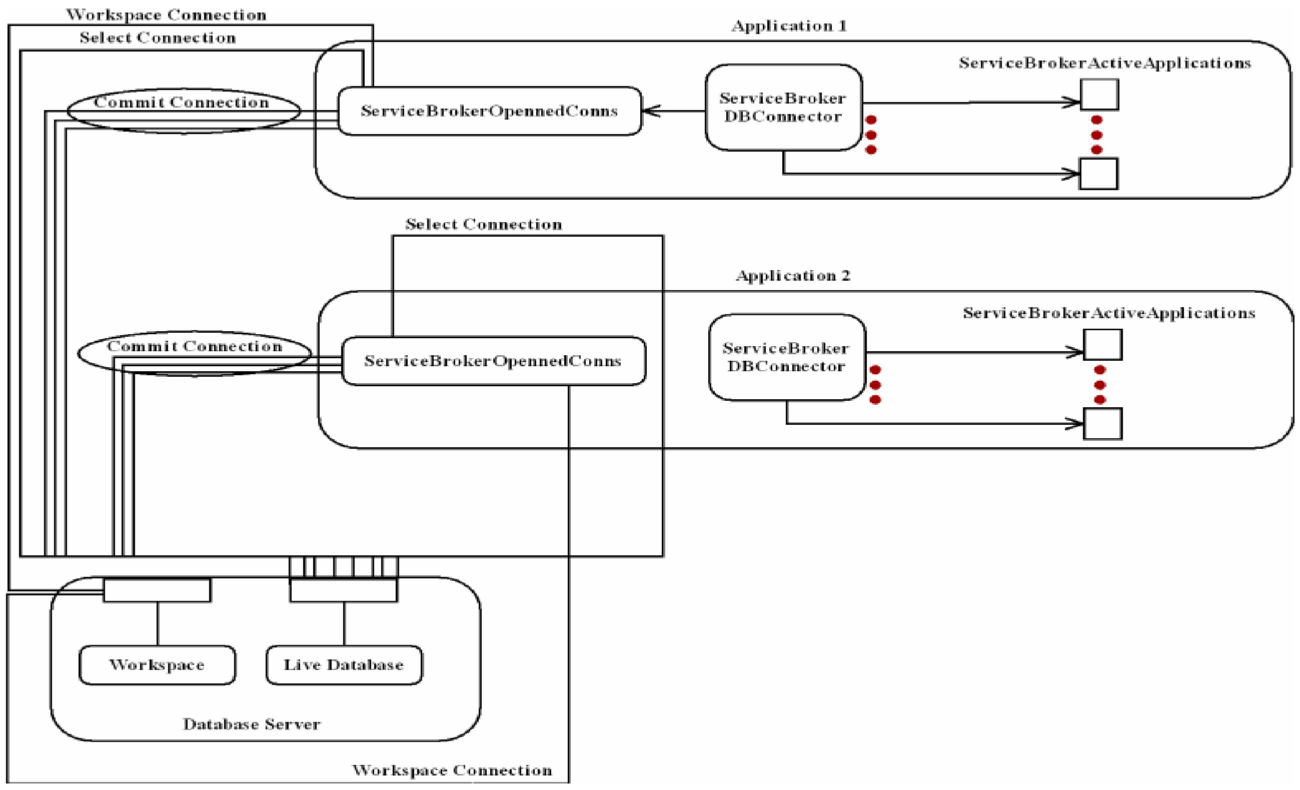

Fig. 5 Connection Management for two running applications

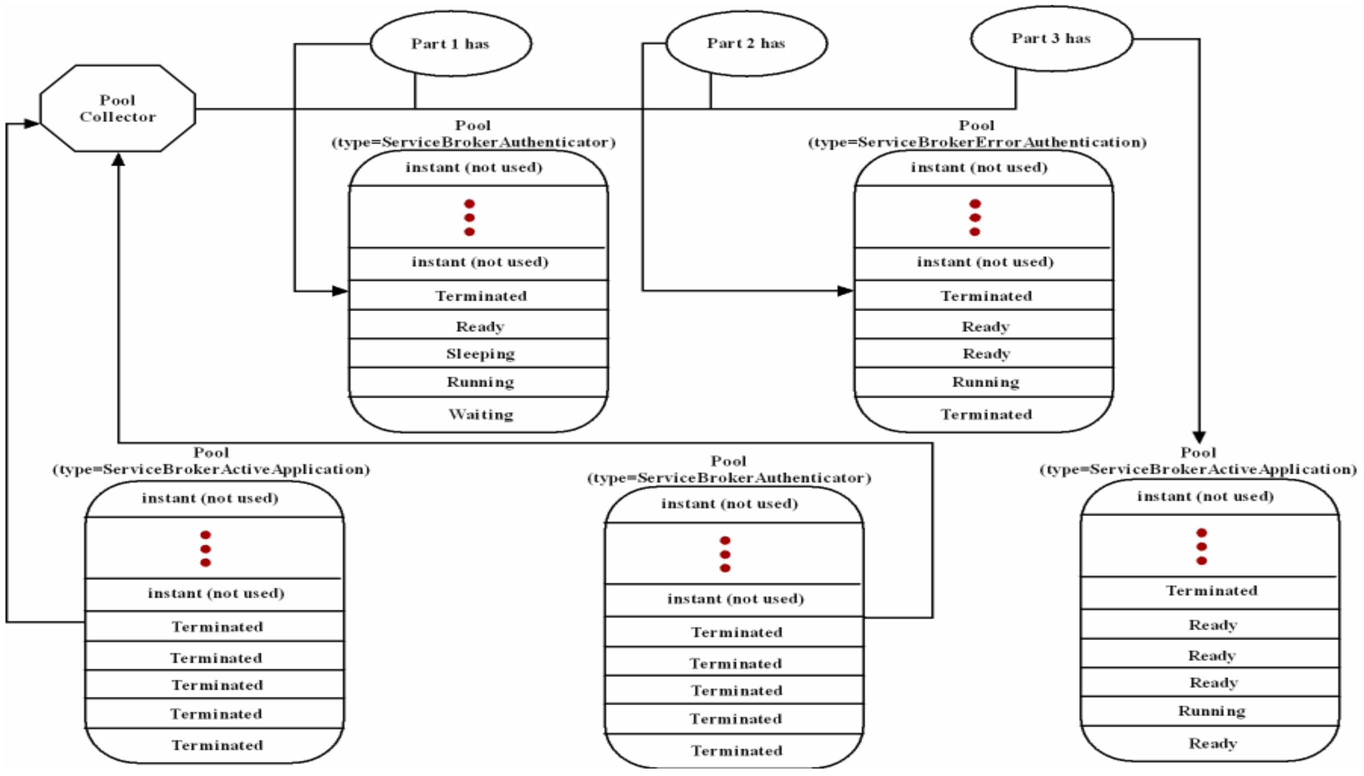

Fig. 6 Pools and Pool Collector

B. Query Optimization

One of the active applications may sometimes try to perform the Select statement while the same Select statement is in progress. In this case, if the system performs the same query, then the system will slow down because of redundant execution.

Therefore, if one of the active applications wishes to perform a Select statement, it uses the parent reference to ServiceBrokerDBConnector to check whether this instruction exists in the object-cache framework. If this instruction does not exist in cache, the system checks if one of the current executing queries has the same demand. If another active application has sent the same query and is waiting for the result, then the query goes to the waiting state and waits for a wakeup. After the Select connection returns the result to the active application's object, it checks if there is any request which is waiting to get the same result. If it finds one, then it calls a Notify function to wake all the waiting threads up. Finally, all waiting threads wake up and get the result from the cache.

Note that the searching process did not consume a lot of time to find whether the same query has been executed twice by two requests simultaneously. So if neither a valid nor a binding query were found in cache, then the time for searching is considered to be wasted. Hence, by using this paradigm, an optimized query can be produced.

VI. Results and Experiments

This section illustrates the results obtained from real-system implementation. First, the proposed system has been applied to Apache web server, and then experiments were made using two phases. The first of which is when the web server does not use a database Service Broker, and the second is when the web server uses a database Service Broker.

A. Experiment's Description

The experiment is supposed to retrieve nearly 64,000 distinct records, by using one of the powerful web servers. These records are retrieved from a schema that holds various tables, which represent the department of an employee issue for the Balqa Applied University. All records in this schema will be retrieved from web server; firstly without using Service Broker and secondly by using Service Broker.

Load balancing and clustering techniques were not applied; [5] [6] [8]. In both phases, the 64,000 distinct records retrieved from 24 queries. Table I illustrates each query with its size.

B. Experiment's Environment

- Web Server: Jakarta [3] (one of Apache projects). This web server applied in Oracle project called (Oracle 9i JDeveloper v.9.0.3.2).
- Database Server: Oracle 9i database [14].
- Operating System :
  - Microsoft Windows XP Professional / Version 2002 / Service Pack 2
  - Intel (R) Pentium (R) M / Processor 1.60 GHz, 1.60 GHz
  - 736 MB of RAM
  - Full cache

C. Experiment's Goal

The experiment's goal is mainly to find out whether the Service Broker improves the web server performance or not. In the first phase, the 64,000 records have been retrieved without using the Service Broker. This phase has been performed 10 times with the same conditions and calculated the average total time. In the second phase, the 64,000 records have been retrieved using the Service Broker. This phase has been performed 10 times with the same conditions, and calculated the average total time. While performing any of the two phases, a set of data manipulation languages has been applied to assure data consistency and concurrency control.

D. Experiment Results without Using Service Broker System

Using BC4J in JDeveloper 9.0.3.2 instead of using the Service Broker system. The average total time needed to retrieve nearly 64,000 distinct records from oracle 9i database is 7 minutes and 15 seconds. The Table II shows the result of an experiment without using Service Broker.

E. Experiment's Result Using Service Broker System

The same previous experiment was performed within the same conditions but using the Service Broker system between front and backend servers this time. The average total time needed to retrieve nearly 64,000 distinct records from oracle9i database using the Database Service Broker was 28 seconds. The Table III shows the results of an experiment using Service Broker.

VII. Conclusion and Future Work

A possible way to improve the performance of a web server is by using a database Service Broker for decreasing the overload on the database backend server was developed. This Service Broker system can be used as a middleware between a front-end web server and a backend server in dynamic pages instead of using APIs to access a database server.

This paper presents a foundation for a middleware that uses a limited number of open connections to a database server. It receives messages from a front-end web server, then it sorts and rewrites these messages and produces a query. After receiving replies from the backend server, the Service Broker sends the results to the front-end and makes a local copy to serve similar requests if possible. Moreover, an object-oriented module for thread management and connection management techniques, that improved the performance of the application by managing memory allocation, thread control, and connection optimization, was designed.

Service Broker defines a strategy for sharing among connections. It also minimizes the context switching, general judgment, a small start up – shut down connection operations and memory management.

An important direction for future work is to apply Multi Query Optimization on the system in order to relief the database server from some of the database management overhead. This requires employing the metadata perspective and query parsing, but consumes a higher processing time.

TABLE I
THE 24 QUERIES THAT HOLD THE 64.000 DISTINCT RECORDS

Query	Number of records in each query	Query	Number of records in each query
1	175	13	3
2	2276	14	26
3	16	15	21
4	33	16	175
5	62	17	64
6	2	18	39
7	7	19	86
8	19	20	2
9	8	21	0
10	2	22	0
11	22	23	60907
12	3	24	254

TABLE II
EXAMPLE OF AN EXPERIMENT RESULTS WITHOUT USING SERVICE BROKER

Query	Time consumed for each query /ms	Total time counter /ms	Query	Time consumed for each query /ms	Total time counter /ms
1	210	9373	13	30	20469
2	10886	20269	14	0	20469
3	10	20279	15	30	20499
4	10	20289	16	10	20509
5	10	20299	17	10	20519
6	10	20309	18	60	20579
7	10	20319	19	81	20660
8	10	20329	20	20	20680
9	10	20339	21	20	20700
10	10	20349	22	10	20710
11	10	20359	23	416498	437208
12	10	20439	24	31	437239

TABLE III
EXAMPLE OF AN EXPERIMENT RESULTS USING SERVICE BROKER

Query	Time consumed for each query /ms	Total time counter /ms	Query	Time consumed for each query /ms	Total time counter /ms
1	241	962	13	220	2644
2	1001	1963	14	10	2654
3	0	1963	15	10	2664
4	10	1973	16	130	2794
5	180	2153	17	10	2804
6	9	2164	18	30	2834
7	10	2174	19	141	2975
8	230	2404	20	10	2985
9	10	2414	21	0	2985
10	0	2414	22	0	2985
11	10	2424	23	24936	27921
12	0	2424	24	10	27931

REFERENCES

[1] Huamin Chen,Prasant Mohapatra, "Using Service Brokers for Accessing Backend Servers for Web Applications",IEEE,the National Science Foundation through the grants CCR-0296070 and ANI-0296034, 2003.

[2] T. Abdelzaher, N. Bhatti, "Web Content Adaption to Improve Server Overload Behavior", in international World Wide Web conference, May 1999.

[3] Apache HTTP Server Project, http://www.apache.org, Mar 2005.

[4] D.M. Dias, W. Kish, R. Mukherjee, R. Tewari, "A scalable and highly available Web server", IEEE Computer Society Int. Conf. Feb. 1996.

[5] X. Zhang, M. Barrientos, J. Chen, and M. Seltzer, "HACC: An Architecture for Cluster-Based Web Servers", in Proceedings of the Third USENIX Windows NT Symposium. July 1999.

[6] V. Cardellini, M. Colajanni, and P. S. Yu, "Load Balancing on Web-server Systems", IEEE Internet Computing, May/June 1999.

[7] R. S. Engelshall, "Balancing your web site. Practical approaches for distributing http traffic", WEB-Techniques, May 1998.

[8] A. lyengar, E. MacNair, T. Nguyen, "An analysis of web server performance", in GLOBECOM, Nov 1997.

[9] D. Mosberger and T. Jin, "httpref: A tool for measuring web server performance", in WISP, ACM, June 1998.

[10] M. Crovella, A. Bestavros, "Self-similarity in World Wide Web traffic: Evidence and possible causes", IEEE/ACM Trans. On Networking, Dec. 1997.

[11] M.F Arlitt, C.L Williamson, "Web server workload characterization: The search for invariants", IEEE/ACM Trans. On Networking, OCTOBER 1997.

[12] K. F. Eustice, T. J. Lehman, A. Morales, M. C. Munson, S. Edlund, and M. Guillen, "A Universal Information Appliance" IBM Systems Journal 38, No. 4, 575–601 (1999).

[13] F. Kitayama, S. Hirose, and K. Kuse, "Dharma: A Framework for Development of WebApplications for Pervasive Terminals—Systems Overview and Application Objects" IPSJ 57th Annual Convention, in Japanese (1998).

[14] http://www.oracle.com, Mar, 2005.

[15] http://msdn.microsoft.com, Mar, 2005.