Dependency Free Distributed Database Caching for Web Applications and Web Services

Hemant Kumar Mehta
School of Computer Science and IT, Devi Ahilya University
Indore, India

Priyesh Kanungo
Patel College of Science & Technology
Indore, India

Manohar Chandwani
Department of Computer Engineering, IET, Devi Ahilya University, Indore, India

ABSTRACT
Rapid growth of technology enabled several newer scientific and commercial applications to be developed, demanding increased computing power. Distributed computing is necessary to satisfy the increasing demand of these applications. Almost every commercial/ scientific application is developed as either web applications or web services or both. The users of such system are increasing exponentially. Generally, these web applications are dynamic in nature. The high demand servers receive thousands of requests in a second. This overloads the database and degrades the overall performance of the applications. In multitier web applications and web services the web based frontend, business logic and databases are deployed over the different servers. To avoid the network delay and to improve the performance of such applications a new distributed hash-based database caching (DHBDC) has already presented [18]. However, to use DHBDC the applications must be modified to access the data from the cache. This paper modifies the DHBDC in such a manner that these applications need not to be upgraded. This new strategy called Dependency Free Distributed Database Caching (DFDDC and is integrated with a custom JDBC driver to achieve the dependency freedom. The prototype of new strategy has been implemented and promising results have been obtained from the prototype.

General Terms
Caching, Web Applications, Performance Enhancements

Keywords
Database, Database Caching, DHT, Multitier Web Applications, Web Service

1. INTRODUCTION
The popular web applications have observed exponential growth in terms of number of requests from different users. Now a days, various kinds of applications are implemented as web applications. These applications vary in the load they generate and the level of usage of the database in the request processing. These web applications use database at various level e.g. (i) simple search based applications, used to display the information stored in the database, (ii) heavy database transaction, sites with heavy commercial transaction involving secure transaction, (iii) web sites with multimedia content and (iv) web services develop to be accessed from the various web application etc.

Different types of web applications generate load from low level to very high level. Web services are web applications that use the hyper text transfer protocol (HTTP) for communication, similar to the web sites. However, the web services differ from the web application in the manner they have been accessed by the clients. The web sites are implemented for human users while the web services are intended to be used by web applications. Web services are not collection of static and dynamic web pages like the web sites, instead, they hosts various functionalities in the form of functions or methods that are accessible through the various HTTP client applications.

The smaller web applications are deployed over the single server running both the web server and database server. The web application with large number of users and requests cannot be effectively hosted on the single server as it may be overloaded while handling large number of requests by including results from the database. To cope up with this problem, the functionalities of these applications are divided among the multiple tiers. Depending on the load on the web application, each function may be deployed on a separate server. In this manner, the specialized hardware may be used for processing. Web applications divided into tiered form is called multitier web application as shown in the Figure 1.

Web application depicted in Figure 1 is a three-tier application. The client sends the request to access some resource on web application. This request is received by a frontend web server. This web server then transfers this request to the application server having the business logic of the application. This application server interacts with the database server to satisfy the user's request. This division of the requests into multiple levels avoids the overloading of the single server. However, the multitier web applications introduces extra load on network bandwidth and introduces the network delay as the frontend server, business logic on application server and database servers are three different systems.

Web caching is a solution for the bandwidth and network delay problems in the multitier applications. Frequently accessed data items are stored in the main memory of the either frontend web server or the application server. The data item requested by the client is first searched in the cache memory. If data is found, it is returned from the cache memory. If data is not available in cache memory then it is searched into the database. There are several issues associated with the cache handling. These issues are selection of the candidate data to be cached, the location of the caching, freshness, consistency of the data etc. The candidate for caching must be the dynamic data that is changing but stable for some time period so that it is meaningful to be cached. Similarly, if the access frequency is too high, the data with short life span may also be cached. Ideally, the data should be cached at the nearest location to the users. However, the data security issues must be considered while selecting the location of the caching.

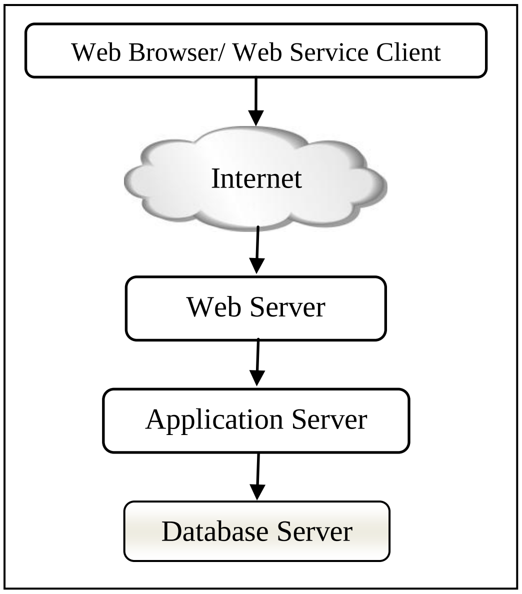

Fig 1: The multitier web applications

The data freshness and consistency are the most challenging task in the caching mechanism. An effective mechanism is required for the identification of the data that have been changed or updated. This process is called cache validation.

The databases used in the multitier application are heavy weight databases and therefore kept away from the application servers. Database caching is the process of the caching of data of the main database server into a light weight database at application server. These light weight databases keep the frequently accessed data at the application server. This implementation of database caching improves the throughput of the web application by avoiding the network overheads. The distributed implementation of database caching can further improve the performance of the web server by handling the bursty load of the users' requests. The distributed database caching can be implemented on web server or application cluster, where several servers are being used for serving users' requests. Cache can be replicated at all the servers in the web cluster, which means that the data is duplicated at all the servers. Another implementation can be division of the data among the servers in the web clusters. To improve the performance of such applications, we have already presented a Distributed Hash-Based Database Caching (DHBDC) [18]. However, the disadvantage of DHBDC is that, before using this scheme, the application must be modified to access the data from the cache. We provide solution to this dependency problem.

In this paper, a new Dependency Free Distributed Database Caching (DFDDC) strategy is presented for the multitier web applications and web services. Section 2 discusses the problem addressed in the paper. Section 3 presents proposed solution. The experimentation and results are described in Section 4. After discussion of related work in Section 5, Section 6 presents concluding remark and future directions.

2. PROBLEM ADDRESSED

The multitier applications divide the responsibility of the web applications into multiple layer called tiers. Each tier is handling a single task. This will improve the performance of each tier, thereby increasing the performance of overall web application. However, this division introduces a new problem related to network delay. The database caching at some stage can reduce the actual database access to certain large extent. The database caching imposes several challenges for successful implementation. Most significant challenges are data invalidation, data expiration, caching granularity, data selection for caching and the location to store the cache.

The data invalidation means the freshness and consistency of the data. Data expiration means purging of unnecessary data from cache memory either based on aging or using some data replacement algorithm. Caching granularity refers to providing different level of caching the data e.g. option to cache complete database table or caching of selected rows and columns of the table. Data selection involves the selection of the data to be cached so that the caching achieves high cache hit rate. The location of the cache is important for the data privacy and security. Proper location of caching will affect the performance of the entire web application. The database is accessed by the business logic stored at application server. Therefore, the database caching can be stored at application server. If request processing does not involve complex business logic then database caching can also be performed at web server level for improved response time. There are several alternatives to the database cache. These are hardware cache available with CPU and caching provided by database venders. However, these alternatives are having limited size for data storage, while multitier web applications require large spaces for caching.

The main memory databases (MMDB) are scalable than hardware cache and faster solution to the caching provided by database venders. MMDB requires efficient data structure to store large set of data required by multitier web applications. The hash table is used for storing the data in the main memory. If the data set is too large then a single hash table may become bottleneck. Distributed main memory database (DMMDB) can be used if the dataset is large and highly demanding to avoid single point of failure and bottleneck problems. The DMMDB keeps the caching records at more than one node in either replicated or division mode. DMMDB requires efficient request-to-server mapping mechanism for better response time.

The caching strategy should be implemented in such a manner that its existence is transparent from the application program and the databases. This can be achieved by modifying the database connectivity driver, in a manner that it first searches the cache for the desired data and if there is a cache miss than actual database is searched. As the database connectivity driver is modified, there is no requirement of modifying application program.

3. THE SOLUTION: DEPENDENCY FREE DISTRIBUTED DATABASE CACHING

The Dependency Free Distributed Database Caching (DFDDC) is presented here as a solution to the problems discussed in the section 2. This solution is implemented using Java based technologies as a prototype. However, the solution can be extended to incorporate all the technologies. This is a generic caching policy that is able to work for any application and can be configured for any database. The larger hash tables are used as database caching mechanism. The hash table is distributed in a server cluster. Here, both the replicated and non replicated models are implemented for data caching where user may select any option as configuration parameter. Centralized caching control is implemented for efficient and better management of the caching [16].

In replication model, the central controller manages the load distribution among the cache servers. In non-replicated model, all caching items are equally divided among the candidate servers. The central controller manages a distributed hash table (DHT) for the mapping of data item to the server containing the cached items [4]. The DFDDC is integrated with a newly developed custom JDBC driver. The custom JDBC driver is implemented so that the existing application program remains unchanged. This JDBC driver connects directly to the cache cluster instead of connecting to the database. If the desired data is not available in the cache, the data is fetched from the database. The JDBC driver also contains the centralized controller of the cache.

Architecture of the DFDDC is depicted in Figure 2. The client request is sent to the frontend server. The request is then transferred to the application server. The application deployed on the server uses the JDBC driver. This driver is replaced with a newly developed custom JDBC driver. This driver uses DHT to determine the cache node possibly having the data. The cache lookup is performed at the cache node. If it is cache-hit, the request is fulfilled from the cache. If cache lookup is cache-miss then the actual database is used to fetch the requested data and the cache is updated with the data. If the cache is full then, the least recently used (LRU) algorithm is used for the cache item replacement otherwise the data is updated in the cache itself [8]. The data expiration is also implemented based on the age or Time-to-live (TTL) and the idle time.

The data update request is fulfilled from the cache level i.e. all the updates are first performed at cache level and later on a synchronization process is executed to finally update the database. An update buffer is maintained for keeping the track of update operation. The synchronization process is executed periodically as well as triggered when the update buffer is full. In this manner, no explicit data invalidation is required in the DFDDC and data consistency and freshness is maintained. All this is effectively manageable because of centralized cache management.

3.1 ALGORITHM OF DFDDC

The algorithm of the DFDDC is presented in this subsection. The user requests are received by application server and the deployed application forwards this request to the custom JDBC driver. DFDDC is implemented in the custom driver, that works as follows:

Algorithm

1. *The custom driver with DFDDC, receive the request from the application server. Go To step 2 if the request is for data searching and. if it is update request, go to step 3.*
2. *Perform the following steps*
    a. *Determine the cache server having the requested data using DHT.*
    b. *Prepare the request and retrieve the data from cache server, if data is not available in cache then cache retrieves the data from database.*
    c. *Send back the data to the application.*
    d. *Update the cache to include the newly fetched data for future requests.*
3. *Perform the following steps*
    a. *Determine the cache server having the requested data using DHT.*
    b. *Send the update request to cache server.*
    c. *Cache server locally updates the data. The final update will be sent to actual database either periodically or when buffer is full of updates.*
    d. *If the data is not available in the cache than actual database is modified and the updated data is retrieved in the cache.*

End of the Algorithm

4. EXPERIMENTAIONS AND ANALYSIS OF RESULTS

Extensive experimentations are performed to test the validity and practicality of proposed approach. The experiments are performed using open source software. The MySQL database is used along with JEE, Apache web server and Apache Tomcat application server [10, 5, 2, 1]. The requests are generated using a load testing tool named as Apache JMeter [7]. Following two subsections describe the experimental setup and analysis of the results obtained.

4.1 EXPERIMENTAL SETUP

The performance of DFDDC is evaluated on the same experimental setup on which DHBDC was tested. The experimental setup is created consisting of seven servers. Separate servers are used as web server, application server and database server. Remaining four servers are used for building cache cluster. The cache central controller uses remote method invocation (RMI) to interact with the servers in cache cluster [15]. All the servers use RedHat Enterprise Linux Advance Server 4.0 operating system. Table 1 summarizes the details of experimental setup.

Table 1. Experimental Setup of Seven Servers

Server	Software	Hardware
Web Server	Apache web server	IBM server having Intel dual Quad Core processor with 4 GB RAM.
Application Server	Apache Tomcat + Custom JDBC Driver	IBM server having Intel dual Quad Core processor with 4 GB RAM.
Database Server	MySQL database	IBM server having Intel dual Quad Core processor with 8 GB RAM.
Caching Cluster (4 Servers)	Java RMI	IBM server having Intel dual Quad Core processor with 8 GB RAM.
Client Machines	JMeter	HP Desktop having Intel Core 2 Due, 2.80 GHz processor with 2GB.

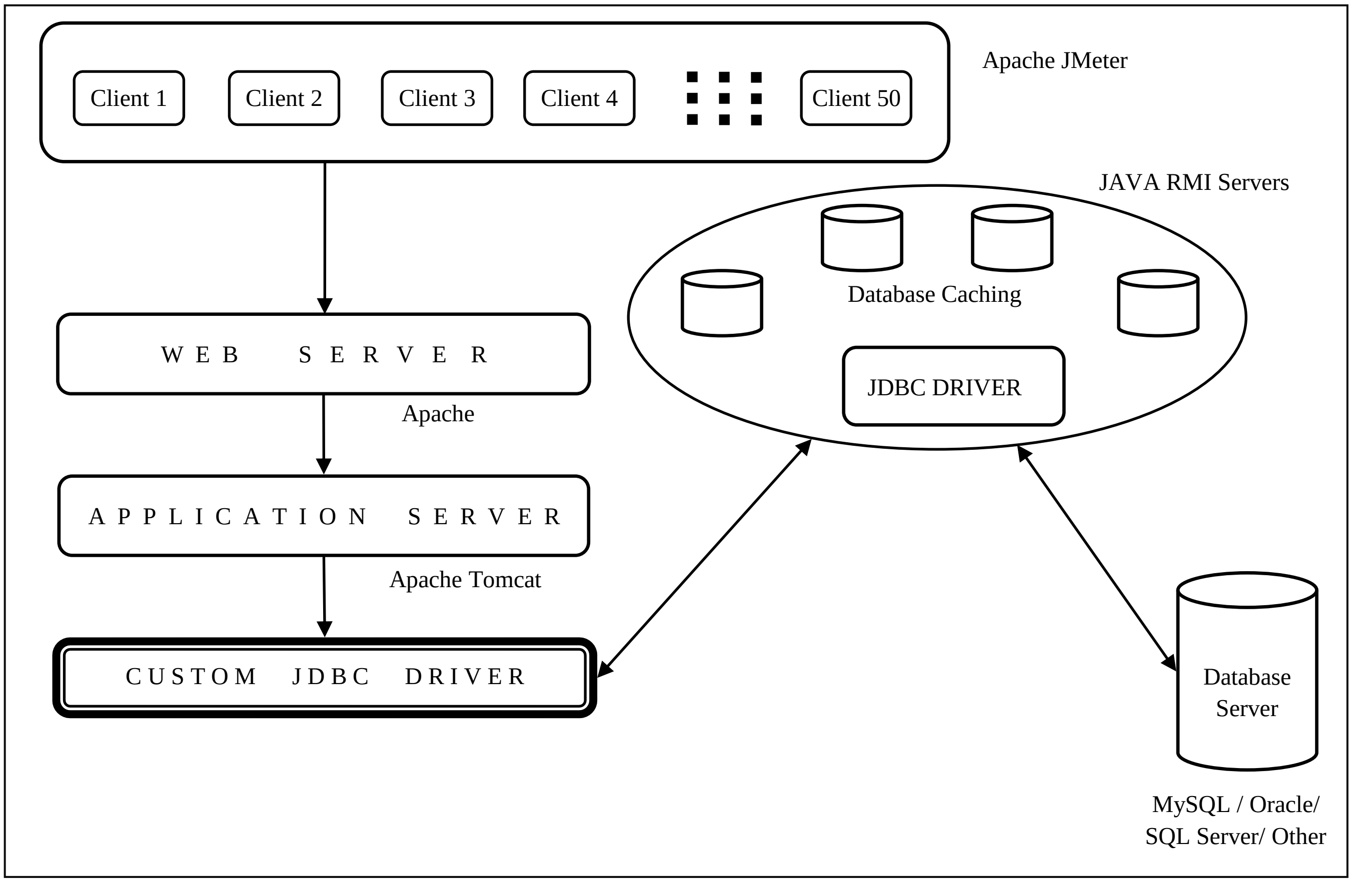

Fig 2: The implementation architecture of DFDDC.

The JMeter is configured to generate requests from 50 different clients (Threads). Each client will generate 1000 request. Total 50,000 requests are generated for testing. The request to fetch the details of the order by joining three tables namely Customer, Order, Order_Line. The searching is performed on the total 1,00,000 orders. Total ten repetition of the same experimentation are performed. The average processing time of all the requests generated for an experiment is used in the graph. The JMeter sends the request to the web server.

The cache is implemented in two ways viz. the distributed and the standalone cache at application server. In case of distributed cache four servers are used in the cache cluster. On the other hand the standalone cache is created at application server only.

A website is tested in this environment. This website is developed using Servlet and JSP interacting with MySQL database. However, the custom driver can be used with any existing databases such as Oracle, SQL server, Derby etc. The static part of the website is deployed on Apache web server. The dynamic part along with business logic is deployed on Apache Tomcat application server. The web site is an online book store deployed over the web server and the application server. The database consists of the following five tables:

- Customer: Table having the details of the customers.
- Country: The table contains country and their economy information.
- Order: The table keeps the details of orders placed by the customers.
- Order_Line: The table contains the detail of the books included in the order.
- Author: The table stores information about the authors of the books.

4.2 ANALYSIS OF RESULTS

The graph in Figure 3 is plotted from the results of the experiments performed. The graph is plotted between three points namely DCH-DFDDC, DCH-DHBDC and DCM. The DCH and DCM represent the values when the distributed cache is used and data hit (DCH) in cache and data miss in cache (DCM) is occurred respectively. The DCH is compared for the two strategies DFDDC and DHBDC. Besides the direct advantage of wider applicability due to the dependency freedom, another advantage is in terms of performance that can be observed from the results. Around 8% performance enhancement is achieved if it is a cache hit in distributed cache as compared to DHBDC. The reason of the performance improvement is the implementation of the custom JDBC driver along with the centralized cache controller. This integration reduces the extra overhead of communication between driver and the cache.

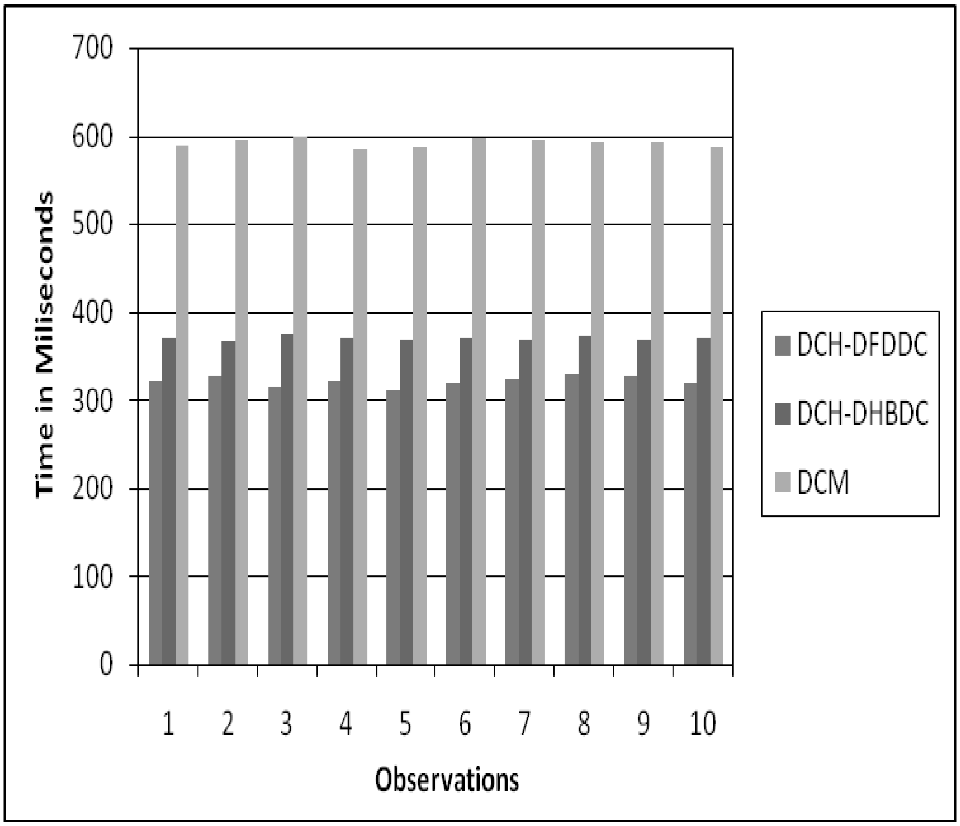

Fig 3: Performance comparison of DHBDC and DFDDC.

5. RELATED WORK

There exists a large body of research work on various caching techniques. However, the focus of this paper is database caching. Few research work and industrial products are available to improve the performance of web-based database applications using database caching. However, most of these databases caching work mainly focused on the specific databases.

Memcached is among the most popular object caching tool. This tool uses large hash tables for caching of the objects. Memcached stores strings and objects from the results of database calls, library calls and rendering of web pages. There is no centralized control in Memcached [9].

Oracle Times Ten In-Memory database is also another popular product providing faster access to the relational databases using the same programming interface. Time ten improves performance as all the data is stored in the memory and no disk based operations are required to access the data. Times ten can optionally be configured to store checkpoint and logging data to the disk [12, 17].

Paul and Fei have presented a distributed architecture for caching with centralized control (DEC3). They have evaluated this architecture using three parameters namely hit ratio, response time and the traffic on backbone link. They demonstrated that the centralized control in this architecture utilizes the caching resource to improve the above parameters. However, they have implemented the caching for the web objects like HTML pages and images [16].

Tolia and Satyanarayanan have implemented hash-based technique for caching of database results to improve the response time and throughput of the web applications. They named the project as Ganesh having two components. The first component of Ganesh is called Ganesh Java JDBC driver that maintains an in-memory cache for the previous query result. The second Ganesh component is a Ganesh proxy that uses original JDBC driver to connect to database. The proxy is also having mechanism to detect the similarity in the query results. In this manner Ganesh do not affect web server, application server and database server at all [11]. Although Ganesh is hash-based but this is not the database caching mechanism. It caches the part response of dynamic web pages.

Larson et al. have developed a mid-tier database caching policy for Microsoft SQL Server database called MTCache. They developed this caching policy to improve the response time and system throughput by sharing some of the load of database server with the cache servers. MTCache can only work with Microsoft SQL Server and not with the other databases. However, the DFDDC is a generic policy that can be configured for any database [13].

DBCache is another proprietary product developed by IBM for their DB2 UDB database. This is also a middle-tier database caching policy [14]. This will support the data access from cache or database or both in a distributed manner. Times-Ten, MTCache and DBCache are proprietary products that can only work with specific databases [3].

6. CONCLUSION

This paper presents a distributed hash-based database caching mechanism for multitier web applications and web services. This is a generic cache policy that can work with any database. The centralized cache control is implemented for better overall performance. The database products are having built-in cache implementation. However, these built-in caching is not a scalable solution while DFDDC presented here is a scalable solution with better performance. DFDDC is having better acceptability in various applications due to the dependency freedom. Extensive experimentations are performed to test the validity of the proposed method. In comparison with DHBDC, around 8% enhancement is observed from the experimentation performed.

In future, testing of DFDDC on real applications and standard benchmark Java Middleware Open Benchmarking (JMOB) is planned [6] to reflect the real-time performance of the proposed strategy.

7. REFERENCES

[1] Apache Tomcat Application Server, http://tomcat.apache.org/.

[2] Apache Web Server, http://www.apache.org/.

[3] Bornhövd, C., Altinel, M., Krishnamurthy, S., Mohan, C., Pirahesh, H. and Reinwald, B. 2003. DBCache: Middle-tier Database Caching for Highly Scalable e-Business Architectures. In Proceedings of ACM SIGMOD International Conference on Management of Data (San Diego, California, USA, June, 2003). SIGMOD’03, 662-662.

[4] Distributed Hash Tables, Available Online at, http://en.wikipedia.org/wiki/Distributed_hash_table.

[5] Java Enterprise Edition, http://java.sun.com/javaee/.

[6] Java Middleware Open Benchmarking (JMOB), http://jmob.ow2.org/jvm.html.

[7] JMeter Load Testing Tool, http://jakarta.apache.org/ jmeter/.

[8] Least Recently Used, Available Online at, http://en.wikipedia.org/wiki/Cache_algorithms#Least_Recently_Used.

[9] Memcached Distributed Memory Cashing System, http://memcached.org/.

[10] MySQL Enterprise Server, http://www.mysql.com.

[11] Tolia, N. and Satyanarayanan, M. 2007. No-Compromise Caching of Dynamic Content from Relational Databases. In Proceedings of 16th International World Wide Web Conference (Banff, Canada, May 2007). WWW’07, 311-320.

[12] Oracle Times Ten In-Memory Database, http://www.oracle.com/database/timesten.html.

[13] Larson, P.A., Goldstein, J. and Zhou J. 2004. MTCache: Transparent Mid-Tier Database Caching in SQL Server. In Proceedings of 20th International Conference on Data Engineering (Boston, Mar-Apr 2004). ICDE’04, 177-189.

[14] Luo, Q., Krishnamurthy, S., Mohan, C., Woo, H., Pirahesh, H., Lindsay B. and Naughton, J. 2002. Middle-tier Database Caching for e-Business. In Proceedings of ACM SIGMOD International Conference on Management of Data, (Madison,WI, Jun 2002). SIGMOD’02, 600-611.

[15] Remote Method Invocation, http://java.sun.com/javase/ technologies/core/basic/rmi/index.jsp.

[16] Paul, S., and Fei, Z. 2001. Distributed Caching with Centralized Control. Computer Communications, Elsevier Press. 24, 2 (Feb 2001), 256-268.

[17] TimesTen Team. 2002 Mid-Tier Caching: The TimesTen Approach. In Proceedings of ACM SIGMOD International Conference on Management of Data (Madison,WI, Jun 2002), SIGMOD’02, 588-593.

[18] Mehta, H., Kanungo, P. and Chandwani M. 2011. Distributed Database Caching for Web Applications and Web Services. Accepted in ACM International Conference and Workshop on Emerging Trends and Technology (Mumbai, February 2011). ICWET 2011, 510-515.