ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

*International Journal of Advanced Research in Computer and Communication Engineering*
*Vol. 3, Issue 10, October 2014*

# A Practical Approach of Application Level Caching for Increase E-Business Performance

**Shaina[1], Mrs. Anshu Kamboj[2]**

Student, CSE, JCDMCOE, Sirsa, India[1]

Asst Professor, CSE, JCDMCOE, Sirsa, India [2]

**Abstract**: Today, the E-Business faces the many problems in terms of efficiency, reliability because of large data has been placed on the servers and the client are using such data at the same and frequently and it also very difficult to manage the E-Business due to these issues. One solution is middle-Tier Database. For E-Commerce websites there are mainly two issues, cost and time. As the numbers of users increase, performance related issues needs to be solved by using caching concepts. In E-Commerce websites dynamic pages are used to provide wider range of interaction than static HTML pages. At the same time, much performance related issues arises by using dynamic page generation technologies because of load placed on server-side resources. For many web applications system support for caching is insufficient. This Paper provides a new way of Application level caching to improve performance of web applications. Concept of shared and unshared caching has been used in proposed method. This paper has been developed the algorithm for demonstrate the actual working of the proposed work.

**Keywords**: Caching, Proxy, Shared, Unshared, Application.

## I. INTRODUCTION

The business related web services are often critical part of infrastructure which is needed for the success of organization in terms of processing time. The application performance benefits can be achieved by the query calculation. The cached objects are often stored in hash table and balanced tree and indexing which can be byte stream, numerical value or by the Strings. The application level cache has been designed with API which allows the developer to manage the cache contents explicitly. There are all features likely to the database which can be update or Modify, Add and Delete. The developers can easily work with the cache by taking the knowledge of application specific Structure when caching of data. There are different approaches for caching which can be Unshared and Shared. [5].

a. Unshared Caching Architecture

In this approach for Caching of Data, the cache is run independently. This feature enable feature of avoid the overhead and complexity of inter-process Communication. There is method which can be used for deploying a cache is to publish as a library which will be as a part of application.

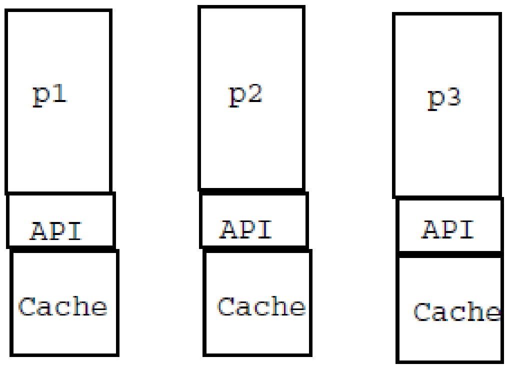


Figure 1: Unshared Architecture

If an application is distributed across multiple processes which need to access a cache, the unshared architecture may be unsatisfactory because it requires each process to manage a different cache. Extra space is required for storing same data in multiple copies if multiple processes need to access it.

b. Shared Caching Architecture

To have a cache operate as a long running process which communicates with multiple processes is known as shared architecture. Using this architecture, it’s only necessary to cache one copy of an object. There are two drawbacks to the shared architecture; one is latency for accessing a cached object may be higher because inter-process communication is required. Second drawback is the if the request rate to a shared cache is high, the cache can become a bottleneck. [2]

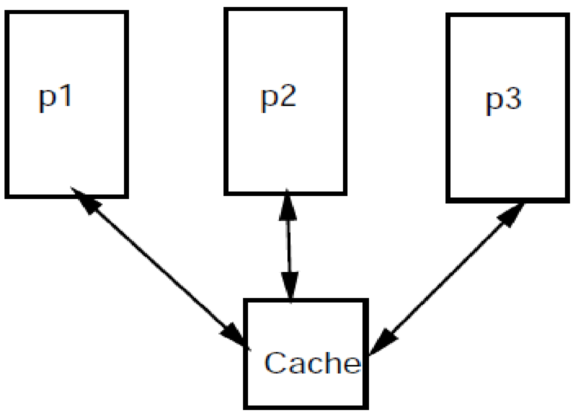


Fig 2 the shared Architecture

The high temporal variability of network traffic results in communication systems that are congested during peak-traffic times and underutilized during off-peak times. One approach to reduce peak traffic is to take advantage of memories distributed across the network (at end users,

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

*International Journal of Advanced Research in Computer and Communication Engineering*
*Vol. 3, Issue 10, October 2014*

servers, routers) to duplicate content. This duplication of content, called content placement or caching, is performed during off-peak hours when network resources are abundant. During peak hours, when network resources are scarce, user requests can then be served from these caches, reducing network congestion. In this manner, caching effectively allows to shift traffic from peak to off-peak hours, thereby smoothing out traffic variability and reducing congestion. From the above discussion, we see that the caching problem consists of two distinct phases. The first phase is the placement phase, which is based solely on the statistics of the user demands. In this phase, the network is not congested, and the main limitation is the size of the cache memories.

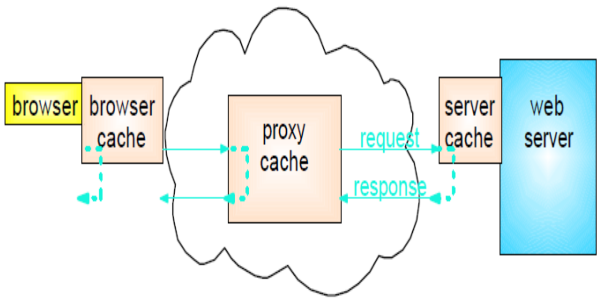


Figure 3: HTTP Caching Today

The second phase is the delivery phase, which is performed once the actual demands of the users have been revealed. In this phase, the network is congested, and the main limitation is the rate required to serve the requested content.

An increasingly large fraction of available bandwidth on the Internet is being used to transfer documents. Strategies for reducing the latency of document access, the network bandwidth demand of document transfers, and the demand on document servers are becoming increasingly important. Techniques that could reduce document latency, network bandwidth demand, and server demand include data caching and replication. However, in contrast to most distributed file systems, document transfer services usually incorporate simple caching strategies, if any, and do not typically provide location transparency.

While techniques based on distributed file systems could be used to improve significantly the performance of document transfer systems, there are a number of advantages to considering caching and replication at the application level, rather than at the file system level. First, application-level caching does not require all users to agree on a common file system; it enables heterogeneous systems to participate easily. Second, and more important, application-level caching allows cache strategies to make use of the higher semantic content available at the application level to exploit such information as document type, user profile, user past history, document content, and organizational boundaries.

## II. LITERATURE REVIEW

Web performance is a key differentiation among content providers. Snafus and slowdowns at major web sites demonstrate the difficulty that companies face trying to scale to a large amount of web traffic. One solution to this problem is to store web content at server-side and edge-caches for fast delivery to the end users. However, for many e-commerce sites, web pages are created dynamically based on the current state of business processes, represented in application servers and databases. Since application servers, databases, web servers, and caches are independent components, there is no efficient mechanism to make changes in the database content reflected to the cached web pages. As a result, most application servers have to mark dynamically generated web pages as non-cacheable. In this paper, we describe the architectural framework of the Cache Portal system for enabling dynamic content caching for database-driven e-commerce sites. We describe techniques for intelligently invalidating dynamically generated web pages in the caches, thereby enabling caching of web pages generated based on database contents. They used some of the most popular components in the industry to illustrate the deployment and applicability of the proposed architecture [7].

E-business sites are increasingly utilizing dynamic web pages since they enable a much wider range of interaction than static HTML pages can provide. Dynamic page generation technologies allow a Web site to generate pages at run-time, based on various parameters. At the same time, however, dynamic page generation technologies have resulted in serious performance problems due to the increased load placed on the server-side infrastructure. Consequently, end users experience increased response times. According to recent research [1], 40% of the total page delivery delay experienced by end users can be attributed to server-side latency. As server-side techniques such as dynamic page generation technologies become more widespread, this percentage will only increase. There has been very little work so far to address the delays associated with dynamic page generation. One proposed approach is to cache entire pages of dynamically generated content (e.g., [3, 5]). However, caching dynamically generated pages in this manner is infeasible, since two calls to the same script with the same input parameters are not guaranteed to produce the same output [8].

While scaling up to the enormous and growing Internet population with unpredictable usage patterns, E-commerce applications face severe challenges in cost and manageability, especially for database servers that are deployed as those applications' back ends in a multi-tier configuration. Middle-tier database caching is one solution to this problem. In this paper, author presented a simple extension to the existing federated features in DB2 UDB, which enables a regular DB2 instance to become a DBCache without any application modification. On deployment of a DBCache at an application server, arbitrary SQL statements generated from the unchanged application that are intended for a backend database server, can be answered: at the cache, at the backend database server, or at both locations in a distributed manner. The factors that determine the distribution of workload include the SQL

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 10, October 2014

statement type, the cache content, the application requirement on data freshness, and cost-based optimization at the cache. Author have developed a research prototype of DBCache, and conducted an extensive set of experiments with an E-Commerce benchmark to show the benefits of this approach and illustrate tradeoffs in caching considerations [9].

Caching is a technique to reduce peak traffic rates by prefetching popular content into memories at the end users. Conventionally, these memories are used to deliver requested content in part from a locally cached copy rather than through the network. The gain offered by this approach, which they term local caching gain, depends on the local cache size (i.e, the memory available at each individual user). In this paper, they introduced and exploit a second, global, caching gain not utilized by conventional caching schemes. This gain depends on the aggregate global cache size (i.e., the cumulative memory available at all users), even though there is no cooperation among the users. To evaluate and isolate these two gains, they introduced an information-theoretic formulation of the caching problem focusing on its basic structure. For this setting, they proposed a novel coded caching scheme that exploits both local and global caching gains, leading to a multiplicative improvement in the peak rate compared to previously known schemes. In particular, the improvement can be on the order of the number of users in the network. Moreover, they argue that the performance of the proposed scheme is within a constant factor of the information-theoretic optimum for all values of the problem parameters [10].

Author considered a network consisting of a file server connected through a shared link to a number of users, each equipped with a cache. Knowing the popularity distribution of the files in the database, the goal is to optimally populate the caches such as to minimize the expected load of the shared link. For a single cache, it is well known that storing the most popular files is optimal in this setting. However, they show here that this is no longer the case for multiple caches. Indeed, caching only the most popular files can be highly suboptimal. Instead, a fundamentally different approach is needed, in which the cache contents are used as side information for coded communication over the shared link. Author proposed such a coded caching scheme and proves that it is close to optimal [11].

## III.PROPOSED METHODOLOGY

While using a full-fledged database engine for middle-tier database caching much research question arises.
The answers to some of them affect the relevance of others. In decreasing order of importance, these are.
1. What are the performance issues in e-Business applications, or in other words, and does the right problems are addressed by us focusing on database caching?
2. Will performance be acceptable using a commercial DBMS as a middle-tier data cache? Features such as transactional semantics, consistency, and recovery come with some operating cost. What features can be dispensed with in such an situation?
3. What database caching schemes are suitable for e-Commerce applications?
4. How can a database caching scheme be implemented in a business database engine and how does it perform under practical e-Commerce workloads?
5. What is the consequence of running a database server in the same computer as an application server?
6. How these results can be generalized to other kinds of web applications?
For middle-tier database caching, using a general-purpose industrial-strength DBMS is especially attractive to e- Businesses. This is mainly due to crucial business requirements such as reliability, scalability and manageability. For instance, an industrial-strength DBMS, provides a variety of tools for application development and closely tracks SQL enhancements. More importantly, transactional support, multiple consistency levels, and efficient recovery services is provided by it. Finally, an ideal cache should be transparent to the application that uses it, and with a special-purpose solution it is very difficult to achieve.

**Objectives:**
1. Remove disadvantages of Shared and Unshared architecture of caching.
2. Design a new Architecture, Hybrid architecture by combining both caching architectures.
3. Design an Algorithm which will handle problem of same data fetching process by maintaining search keywords and counts.
4. Remove caching Memory to store new Data.

**Work Flow of proposed Method:**

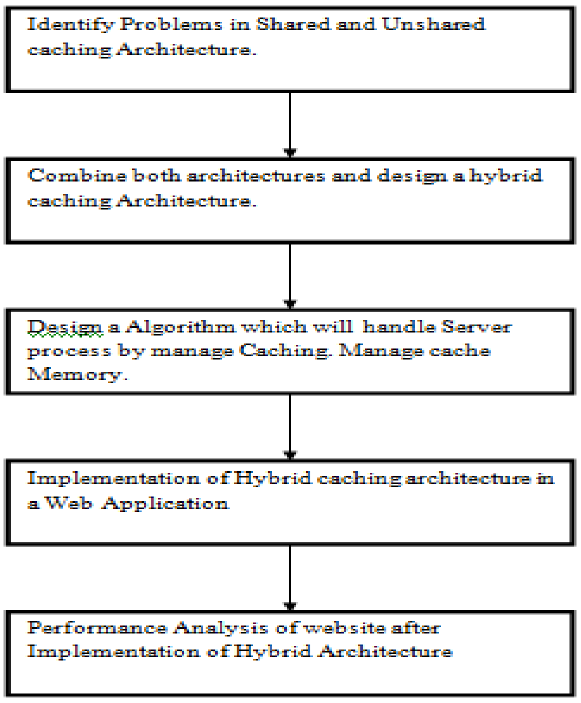
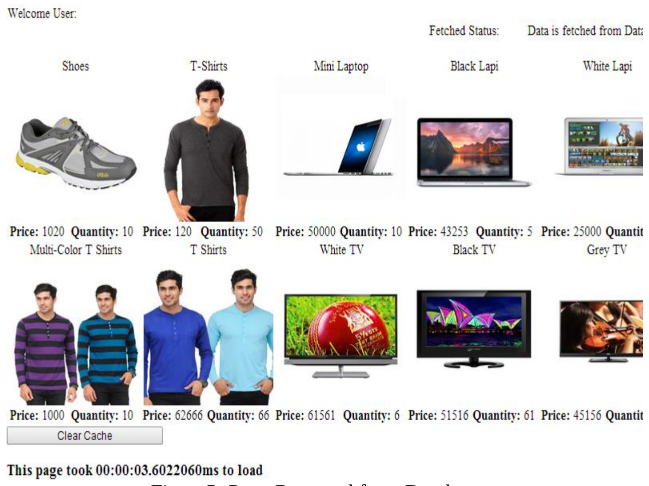

Identify Problems in Shared and Unshared caching Architecture.
Combine both architectures and design a hybrid caching Architecture.
Design a Algorithm which will handle Server process by manage Caching. Manage cache Memory.
Implementation of Hybrid caching architecture in a Web Application
Performance Analysis of website after Implementation of Hybrid Architecture

Fig 4 Flow Chart

## IV. RESULTS

Our Hybrid Caching solution utilizes a application-level caching approach which focuses on re-using Shared/Unshared Cache fragments of dynamic pages. An array consist keywords searched by users with searched count. When count value cross threshold value, a cache will be generated and index value stored. It's shared caching as one cache of 10 tables is used to store frequent search items. Large data tables can be stored in unshared cache. In this case, if we have 2 large pages, we can store complete page in a dedicated cache called unshared cache. It is unshared because other page data can overwrite memory which will destroy page integrity. In our solution, for frequent search we will use shared cache. 10 tables is exists in 1 cache. For large pages like Home page, we will store complete data in unshared cache. So use of both type of caching scheme according to task will improve performance of website. A log file is maintained to measure it's performance. Each time server connects with database, connection time is saved in this file. We will confirm cache working by checking this log file. During data returned by cache, time shouldn't exist in this log file as it only contains time when server connects with database.


Welcome User:

Fetched Status: Data is fetched from Data

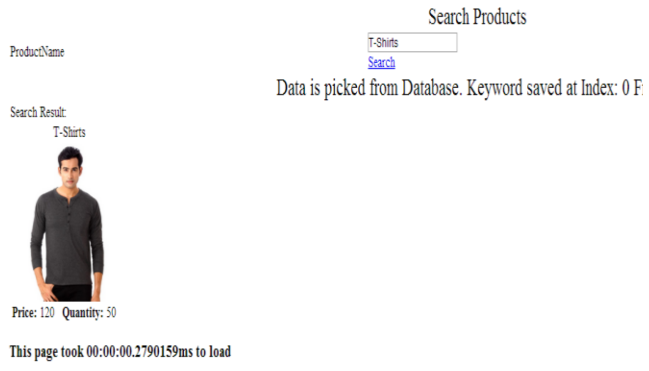
| Shoes | T-Shirts | Mini Laptop | Black Lapi | White Lapi |
|---|---|---|---|---|
| Price: 1020 Quantity: 10 | Price: 120 Quantity: 50 | Price: 50000 Quantity: 10 | Price: 43253 Quantity: 5 | Price: 25000 Quantit |
| Multi-Color T Shirts | T Shirts | White TV | Black TV | Grey TV |
| Price: 1000 Quantity: 10 | Price: 62666 Quantity: 66 | Price: 61561 Quantity: 6 | Price: 51516 Quantity: 61 | Price: 45156 Quantit |

Clear Cache

**This page took 00:00:03.6022060ms to load**

Figure5: Page Returned from Database

The result when use the caching concept is:


Search Products

ProductName T-Shirts

Search

Data is picked from Database. Keyword saved at Index: 0 F

Search Result:

T-Shirts

Price: 120 Quantity: 50

**This page took 00:00:00.2790159ms to load**

Figure6: Search Result returned from Database

Keyword is stored at index 0 with search frequency 1. Page tool 0.2790159ms

Page load time is shown in every page.

Table 1: Performance Improvement using Application Level Caching

| Cache Type | Event | Time(ms) | Returned from |
|---|---|---|---|
| Unshared | Page Load | 3.6022060 | Database |
| Unshared | Page Load | 0.0020001 | Cache |
| Shared | Search | 0.2790159 | Database |
| Shared | Search | 0.0030002 | Cache |

We can see difference of page load time between data returned from database and cache.

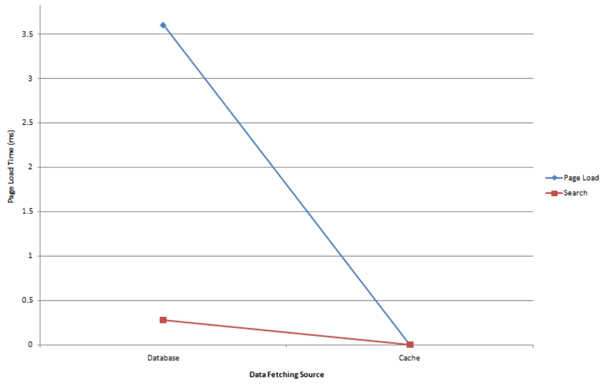


Figure7: Search Result returned from Database

It is clear from the graph that cache has lees time of execution as compare to database fetching as it has been reduced the roundtrip from database.

## V. CONCLUSION AND FUTURE WORK

It's clear from results that caching is effective for large number of user requests and same kind of searching. Site performance can be increased using this caching scheme.

A web application has been designed in Asp.net with caching management algorithm. Results are measured by running it in local system. This site can be deployed on server.

Future work includes extending the Caching Scheme and Algorithm to handle special SQL data types, statements, and user defined functions. Investigating of alternatives for handling frequent database updates. Usability enhancements, such as cache performance monitoring, and dynamic identification of candidate tables for caching are important directions for us to pursue.

## REFERNCES

[1] Jim Challenger, Arun Iyengar, and Paul Dantzig,1999, A Scalable System for Consistently Caching Dynamic Web Data.

[2] Yeol Song, 2000, "Database Design for Real-World E-Commerce Systems", IEEE

[3] K. Johnson, J. Carr, M. Day, and M. Kaashoek. 2000, The measured performance of content distribution networks. In 5th Int. Web Caching and Content Delivery Workshop, Lisbon, Portugal.

[4] B. Krishnamurthy and C. Wills.2000, Analyzing factors that in ence end-to-end web performance. In International World Wide Web Conference.

[5] A. Labrinidis and N. Roussopoulos.2000, WebView Materialization. In ACM SIGMO.

[6] S. Paul and Z. Fei. 2000. Distributed caching with centralized control. In 5th Int. Web Caching and Content Delivery Workshop, Portugal.

[7] K. Selçuk Candan, Wen-Syan Li, Qiong Luo, Wang-Pin Hsiung, and Divyakant Agrawal. Enabling Dynamic Content Caching for Database-Driven Web Sites. Proc. ACM SIGMOD International Conference on Management of Data, Santa Barbara, May 2001.

[8] Anindya Datta, Kaushik Dutta, Helen Thomas, Debra VanderMeer,2001, A Comparative Study of Alternative Middle

Tier Caching Solutions to Support Dynamic Web Content Acceleration

[9] Qiong Luo, Middle-Tier Database Caching for e-Business, 2002.

[10] Mohammad Ali Maddah-Ali and Urs Niesen,2014,Fundamental Limits of Caching.

[11] Urs Niesen and Mohammad Ali Maddah-Ali, 2014,Coded Caching with Nonuniform Demands.