Abstract: The majority of applications served today are generated dynamically, usually by an application server querying a back-end database. As dynamic content becomes increasingly dominant, it becomes an important research topic as how the edge resources such as client-side proxies, which are otherwise underutilized for such content, can be put into use to enhance the scalability of dynamic content serving in large application servers are offloaded to front-end nodes, called edge servers. The improvement from such application offloading is marginal, however, if data is still fetched from the origin database system. To further improve scalability and cut response times, data must be effectively cached on such edge servers. The scale of deployment of edge servers and the rising costs of their administration demand that such caches be self-managing and adaptive. In this paper, we describe DBProxy, an edge-of-network semantic data cache for web applications. DBProxy is designed to adapt to changes in the workload in a transparent and graceful fashion by caching a large number of overlapping and dynamically changing “materialized views”. New “views” are added automatically while others may be discarded to save space. In this paper, we discuss the challenges of designing and implementing such a dynamic edge data cache, and describe our proposed solutions.

Keywords: Query evaluator, Proxy server, Proxy caching, Replacement algorithm.

I. INTRODUCTION

Many applications are constructed using back-end database systems and provide form-based interface for users to submit queries. We call this kind of system a database server. With the rapid growth of user accesses to the database servers encounter very heavy workloads and produce a growing percentage of network traffic. Caching proxies are today’s main solution to improve performance, share server workload, and reduce wide area network traffic. However, queries and responses of database servers are not cacheable by existing proxies, which cache only static files. This motivates us to investigate the problem of how to answer queries efficiently at a proxy. In this paper, we propose a new collaboration scheme between an active proxy (an experimental enhanced proxy server) and a database server [1]. In our approach, the database server passes a simple query processing ability to the proxy when needed. The proxy can then not only answer queries that are an exact match to cached queries, but also queries whose results are contained in the cached results of more general queries. In turn, this reduces network traffic as well as the load on the server, which allows the system to scale with the addition of multiple proxies. We found that offloading and caching at edge proxy servers achieves significant advantages without pulling databases out near the client. Our results show that, under typical user browsing patterns and network conditions, two to three folds of latency reduction can be achieved [2]. Furthermore, more than 70 percent of server requests are filtered at the proxies, resulting in significant server load reduction. Interestingly, this benefit can be achieved largely by simply caching dynamic page fragments and composing the page at the proxy.

II. LITERATURE REVIEW & RELATED WORK

DBProxy minimizes redundant storage and provides lag consistency with update propagation. Database caching schemes, using full or partial table replication have been being proposed [4]. DBProxy is similar, in concept, to the work on semantic caching but it supports consistency and differs in the implementation approach. Semantic caches have been proposed in client-server database systems. The caching of query results has also been proposed for specific applications, such as high-volume major event applications [8]. A simplified form of semantic caching targeting web workloads and using queries expressed through HTML forms has been recently proposed [7] [10]. The semantic cache proposal, however, only handles read-only databases [11]. The importance of database scalability over the application has prompted much industry interest in data caching and distribution. Furthermore, most of them require application server modifications and explicitly bundle the data caching and application distribution logic. DBProxy, on the other hand, is self-managing and does not require any application or database modification as it is bundled as a JDBC driver [1] [2]. Much previous work also exists in the area of query containment and equivalence. The algorithms used for containment checking within DBProxy are based on extensions of previous work in this
area. Earlier work on database caching investigated predicate-based schemes and views to answer queries [12] [13]. Previous work in the area of materialized view routing also describes techniques for matching and containment [14] [15] [1]. DBProxy differs from the materialized view approach in relying only on the query stream to decide on cache population and replacement. DBProxy stores views in common tables and dynamically decides the views that are to be cached or replaced. Consistency management in most materialized view approaches does not scale to a large number of views, while the update propagations on the common-table used by DBProxy simplifies scalability.

III. ANALYSIS OF PROBLEM

There are many applications which are based on the web base proxy server. This application has some of the limitations which we are overcoming in our work.

- The existing applications are mostly developed for the web server.
- One major limitation of current caching proxy server is the lack of collaboration with original content providers. These cause a large amount of traffic, which is not cache by proxy servers.
- There is a large waiting time for the client request response since we are using the index base searching technique.
- The algorithm used were different by which the proxy to need a large time to locate the data.

There are many ways that our work can be extended. We are also using other query caching schemes and cache replacement policies in this framework. There are many advantages which we can get by introducing the new system. The following are the advantages which we will get from our application

- It will minimize the request service delay associated with obtaining query since we are using the hashing technique.
- Here we are using the hash table algorithm so that by using a key it can perform a quick search at each level.
- We have a place replacement manager which is having various different algorithms which are been use in it which is beneficial for different applications to clients.
- Load reduction is the main criteria for developing this application, as we are using the proxy server which reduces the large amount of load of the server. This enables the proxy server to share the database server workload as well as to reduce the network traffic.

IV. PROPOSED WORK AND OBJECTIVES

The architecture of DBProxy assumes that the application components are running on the edge server. The edge server receives client requests and processes them locally; passing requests for dynamic content to application components which in turn access the database through a JDBC driver. The JDBC driver manages remote connections from the edge server to the back-end database server, and simplifies application data access by buffering result sets and allowing scrolling and updates to be performed on them [1]. DBProxy is implemented as a JDBC driver which is loaded by edge applications. It therefore transparently intercepts application SQL calls and determines if they can be satisfied from the local cache.

4.1 Query Evaluator

As shown in above Fig, (a) the cache functionality is contained in several components. The query evaluator is the core module in DBProxy and contains the caching logic. It determines whether an access is a hit or a miss by invoking a query matching module which takes the query constraint and its other clauses as arguments [1]. The query evaluator also decides whether the results returned by the back-end on a miss should be inserted in the cache. It rewrites the queries that miss in the cache before passing them to the back-end to prefetch data and improve cache performance. The resource manager maintains statistics about hit rates and response times, and adapts cache contents and configuration parameters accordingly. DBProxy employs novel techniques for data storage, query matching, consistency maintenance, and cache replacement to achieve the desired efficiency.

4.2 Cache Repository

Data in a DBProxy edge cache is stored persistently in a local stand-alone database. The contents of the edge cache are described by a cache index containing the list of queries. To achieve space efficiency, data is stored in common-schema tables whenever possible such that multiple query results share the same physical storage [1] [4]. Queries over the same base table are stored in a single, usually partially populated, cached copy of the base table.

4.3 Query matching

To handle a large and varying set of cached views, the query matching engine of DBProxy must be highly optimized to ensure a fast response time for hits [1]. Cached queries in DBProxy are organized according to a multi-level index. The first level of the index is the database schema. The second level is the table or list of tables accessed by the query. Finally, the third level of the index contains the columns named in any of the query’s
clauses. Hash tables are used to allow quick search of each level [2] [9]. A new query received by DBProxy is parsed into its constituent clauses.

4.4 Cache replacement
To manage limited space resources on the edge, DBProxy relies on a background garbage-collection process which evicts unused data from the cache safely, while preserving data consistency [1] [4]. The consistency protocol requires that all inserts, updates and deletes performed at the origin site to any cached table be propagated to the edge cache. Consequently, the cache can contain data that does not belong to any cached query. Furthermore, data that is not effectively used must be evicted to limit space overhead and optimize the usage of usually limited edge resources [12]. Specifically, the goal of cache replacement is to maximize the benefit of the cache for a limited amount of available space. In contrast to traditional replacement of files and memory pages, the underlying rows can be shared across multiple queries, complicating the task of query eviction.

Objective
The main objectives are to reduce client response time, network traffic, and server load caused by surges of high volume of requests over wide-area links. Most work focuses on how to support dynamic content caching on server side [3] [4] some others also extend their cache to the network edge and show better performance results.

ACKNOWLEDGMENT
I am grateful to all my lecturers and friends for inspiring me to do my dissertation with immense dedication. With great pleasure and gratefulness, I extend my deep sense of gratitude to Associate Prof. D.M. Dakhane Sir for giving me an opportunity to accomplish my work under his guidance and to increase my knowledge.
Thank You.

REFERENCES