Journal of Web Engineering, Vol.9, No.2 (2010) 116-131

SCHEMA-BASED CACHE VALIDATION OF DYNAMIC CONTENT TO IMPROVE QUERY PERFORMANCE OF WEB SERVICES

A. RAGHUNATHAN
Bharat Heavy Electricals Limited, Trichy, India
raghu@bheltry.co.in

K. MURUGESAN
National Institute of Technology, Trichy, India
murugu@nitt.edu

Received June 19, 2008
Revised January 7, 2010

Web services play a crucial role in e-business, providing application integration within and across enterprises and platforms. Hence there is an increasing need to make web services more efficient and perform better. Caching of data is a vital factor in improving the QoS and query performance of web-based applications. Invalidation mechanisms are used to refresh cache when accessing dynamic data from backend data sources. Time or expiry-based cache validation is suitable for enterprise applications where the data does not change very frequently. However, existing expiry-based caching solutions act at the URL/query level, thus increasing access to the data source and hence the response time. In this paper, we propose a time-based caching technique based on the schema of the data source. Our method performs cache validation at the levels of tables and columns, thus minimizing database access. Moreover, the column level granularity avoids database visits for queries that do not access expired columns. We have used simulations to test our design and the results show a significant improvement in reduction of database accesses for web applications thereby reducing bandwidth usage, server load and network traffic.

Key words: Web services, QoS, performance, caching, database, schema, queries.
Communicated by: B. White & Y. Deshpande

1 Introduction

Web services technology plays a crucial role in e-business today, providing application integration across enterprises and platforms. Quality of Service (QoS) is a significant factor in the performance of web applications [10] and there is an increasing need to improve the efficiency, performance and reliability of web services. One of the key QoS aspects of web services is response time to user requests. Several techniques could be used for the purpose, both at the service level and server level [6]. Techniques used for improving web server performance include caching [1]. Caching of data is a critical performance factor in web-based applications. Caches help to effectively answer queries from locally saved copies instead of fetching them all the time from the originating source, thereby minimizing access time, bandwidth usage, server load and network traffic [31]. Caching can be done at several stages – browser, proxy server, web server or database server. Cache settings are generally made in HTTP headers [7,16]. However, these are effective only for static content like web pages or documents. Web applications also access data from dynamic backend data sources, such as RDBMS and XML, through queries. The tricky problem with caching of such dynamic data is consistency between cache and the database, as queries would have to get the latest content from the data source. However, it is known that not all the data used in e-Business applications vary frequently. Data such as

product catalogs, vendor and customer information do not change for a long time, and, even when they do, it may not be required to provide the updates immediately for viewing. Hence it would make sense to provide the data from a local cached copy, although a bit stale, rather than from the original data source every time. This would save database access time for web applications, which is quite expensive due to network traffic and latency.

In e-business web sites the queries that fetch dynamic content from a backend data source are usually fired in the form of URLs provided on the web sites with or without parameters either as hyperlinks or as forms submitted by users. The web (application) server passes on the queries to the backend database, fetches the query results and stores the cacheable results in its cache [17]. It then formats the results as HTML web pages and returns them to the client which displays them on the browser. When the same query is fired by the client, the (web) application server tries to return query results from the cache, after checking that the data has not changed meanwhile at the database. In case the data has changed, the server executes the query again at the backend and fetches the fresh results into the cache and returns them to the client. A typical web cache architecture is depicted in Figure 1.

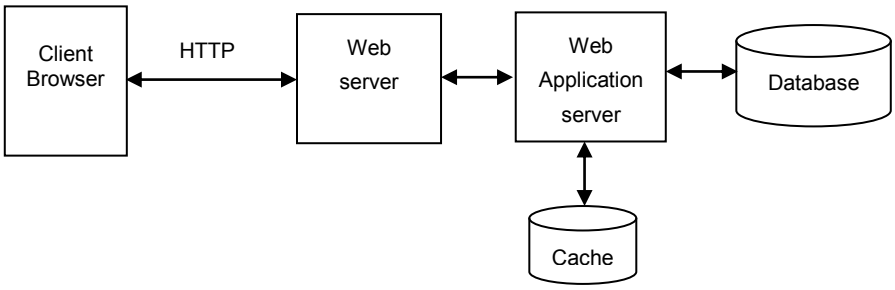

Figure 1 Web cache architecture.

1.1 Cache Validation Methods

In caching dynamic data the cache contents have to be invalidated when the backend data changes. This cache invalidation is generally done through two methods – automatic and expiry-based.

In an automatic invalidation method, whenever the data changes, the database server sends a message to the application server to invalidate the cache portion that contains the changed data. This method is usually implemented through database triggers. While this method ensures cache currency, its implementation using trigger-like mechanisms is expensive and best reserved only for very frequently changing data.

In an expiry or time-based method, a validity time interval is specified for each query or URL whose results are cached. Upon expiry of this time the cache entry for this query is invalidated. When the query is fired again, the web server checks the data source to see if the data has changed, a process called revalidation. If the data has not changed, the server returns the existing cached results to the client without re-executing the query. If the data has changed, the server re-executes the query at the backend and reloads the contents into the cache. Expiration settings give more control over and flexibility in presenting data to the web clients since they interact with the data source only when required. This method is suitable for more predictable, less frequently changing data such as product information in e-business applications.

A query accessing an RDMS contains references to one or more tables and their columns along with optional clauses for retrieving select rows and columns. On a web site hosting database applications, each query application or URL is invoked repeatedly. When the same query is fired several times, the web server returns the cached results for that query, after ensuring consistency of the cached data with the data source. Thus the existing solutions perform cache validation on a URL or application basis, and tend to access the database for every URL or query, which increases the response time. In practice, however, several applications query a common database table, retrieving

multiple columns using various selection criteria. In such cases, it would be desirable to perform cache validation on a table/column basis so that all queries based on that table/column are validated. In this way, the database is visited only once for each table/column combination rather than for each query, thereby saving access time and increasing web service performance. In this paper, we propose a cache management scheme that combines an expiry-based method with a cache structure that leverages the schema of the data source. Our design uses cache invalidation at both table and column levels, by storing a copy of the schema of the data source in the cache. We used simulations to test our design and the results show a significant reduction in database accesses resulting in improved response times of dynamic e-applications over current schemes.

2 Related Work

Extensive work has been published on web caching in respect of consistency and performance. Commercial products such as Oracle Application Server Web Cache 10g and IBM WebSphere are also available. Caching of pages is done at various levels – proxy, mid-tier (web server) or database (called backend). A variety of techniques have been proposed for caching dynamic web pages at all these levels. A summary of these is given in [25]. An active query caching scheme [21] is proposed for proxy servers accessing database backends by using a query applet which runs in the proxy and helps to process queries on the cache contents, thus increasing cache hits and reducing accesses to web and database servers. Extension of active caching to HTML form-based queries is discussed in [22] based on semantic caching [11]. A proxy caching technique is described in [14] where both content and layout can be dynamic, with significant reductions in bandwidths and response times. DBProxy [3,4], an edge semantic data cache for web applications, combines a flexible query answering technique with efficient techniques for storage and cache replacement. Middle-tier database caching has also evolved as an effective approach to improve scalability and performance. A scheme for database caching for database2 relational database, known as DBCache, has been described in [20,2] for web application servers. This caches entire or part of tables or materialized views based on declarative specification, with updates being propagated through asynchronous data replication. A similar product has been developed in SQL server called MTCache [19] whose caching is transparent to applications. The cached data is maintained as a collection of materialized views. In order to reduce delays in caching entire dynamically generated pages, Datta [13] has proposed identifying and caching fragments of them, so that the static part in those pages can be cached, with the dynamic part alone being replaced. Kossman [18] presents an integrated approach to caching with query optimization, which helps to generate optimal query execution plans leading to long-term caching decisions. The Weave System [30] offers customizable caching at different levels. However, it uses a scripting language for declarative specifications of caching rules, and does not specify methods for cache validation on expiry but only cache replacement. A middleware layer that uses hashing techniques to detect similarities with previous cached query results is described in [29]. It thus seeks to avoid transmissions of redundant fragments, thus improving performance.

For less predictable, more frequently changing content, consistency is usually maintained between cache and database by mapping and data update propagation. A Data Update Propagation algorithm is used in [9] to identify changed and obsolete pages in the cache for replacement. Using Object dependence graphs, a dependency mapping is maintained between cacheable data items and the corresponding cached pages. Database triggers are invoked whenever changes are made to database which in turn propagate the changes to the cache. Triggers help to synchronize data updates with generated web pages. The main drawback of this technique is use of request URL to identify pages in cache for retrieval. But the same URL can produce different pages of dynamic output, and this may lead to retrieval of incorrect pages. Also, this uses proxy caching of full HTML pages, and this may incur a lot of space overhead. Degenaro [12] improved the techniques to selectively invalidate cache objects in case of database updates. Candan [8] use sniffer and invalidator modules to invalidate stale cached pages. Sniffer creates a mapping between the web pages identified by URLs and the underlying queries, and invalidator discards the cached pages dependent on queries affected by updates. As triggers used to sense updates introduce a lot of load on the database, the authors use off-line policies

in order to select the query types and URLs for invalidation. The DBCache [20] also uses triggers on their database to track changes. Database change notifications are also used to invalidate the cache in case of updates, and Galindo-Legaria [15] describe the mechanism implemented in SQL server 2005 to communicate to the application through subscriptions regarding database changes for taking action for invalidation. Subscriptions are also used by DBProxy [4] for synchronizing cache with database for consistency. Amza [5] also propose a similar technique for automatic cache invalidation both at table and column levels with a dependency mapping built between database items and cached objects. Of the commercially available web application servers, OracleAS Web Cache [27] features both time-based and automatic invalidation. However, it does not specify a mechanism for automatically invalidating and updating cached pages upon database changes. IBM WebSphere Application Server's Caching Proxy [28] has very similar features. In all, current time-based web caching solutions use a declarative specification of expiry settings and validation for each web application/URL. However, in our approach, we have attempted to validate the cache based on expiration settings at a table/column level so as to benefit other queries based on the same table/column. Our schema-based technique helps to minimize database access for revalidation.

3 Outline of Existing and Proposed Methods

3.1 Existing Method

In existing time-based caching methods, expiry times are specified for each URL/database query that is invoked by a web application (see Table 1). When the web application server receives a URL/query, it tries to return cached results for the query, provided the query has not expired. If the query has expired, it is validated against the data source to check if the data has changed. If there was no change, the query is revalidated. If the data has changed, then the query is re-executed and new (fresh) results are fetched. The cache is refreshed with the new results which are then returned to the client through the application server. We refer to this common query processing technique as Query-based Caching (See Algorithm A). Thus existing web caches perform validation for each URL on expiry. In practical applications, however, several queries may access the same database table (or materialized view). For each of these queries, the database is accessed during every revalidation and re-execution, contributing to a cumulative increase in the response time. The cache structure for a typical query-based caching method is shown in Table 1.

URL/ Query (Q)	Validity Period (VP)	Query Level Validity flag (V_Q)	Timestamp of last validation (TS)	Query Result (QR)
…				

Table 1 Cache data structure in a typical time-based caching scheme.

Algorithm A: Query-based Caching. Process query and return result.

Input: Query Q, Cache Table 1.

Output: Query Result QR.

Method:
// When a query is received, it is checked to see if it already exists in the cache Table 1.
// A background process traverses the cache entries and, based on the timestamp field *TS*,
// marks any expired queries as *Invalid.* (ie., sets V_Q = False.)
if $Q \in$ Table 1 **then** // query is already cached in Table 1
  if V_Q = True for Q in Table 1 **then** // if query is valid i.e., not expired
    return existing cached QR
  else

```
                //perform revalidation
                if Q-related data has changed in the database then
                        execute Q
                        store new query result into QR
                        return QR
                else
                        // related data has NOT changed in the database
                                return existing cached QR
                end if
                set V_Q = True
                set TS = current time
        end if
else    //new query
        add Q to Table 1
        initialize VP, TS
        execute Q
        store new query result into QR
        return QR
end if
```

3.2 Proposed Method

In our proposed caching method, we define expirations at table/column levels rather than at query levels. This way queries that do not access expired columns need not be revalidated or re-executed. Moreover, any action taken to revalidate a query is propagated to the other cached queries that also access the same tables and columns, thus minimizing access to the data source. To enable this, we maintain a copy of the schema of the data source in the cache. This schema copy contains the names of all the database tables and columns that are accessed by the web site. Expiry settings are set at each table level as well as at each column level, and any revalidation action is taken at these levels. A detailed discussion of our proposed method follows.

4 Design of the Proposed Caching Scheme

4.1 Cache Schema management

For our schema-based cache invalidation scheme, we store in the cache a copy of the objects in the database schema. Table 2 stores table names (including materialized view names), validity periods for each table, a table level validity flag (V_T), an aggregate column validity flag (V_C) and the timestamp of the last revalidation / re-execution of any of the queries. The flags are initialized to True, and are then set to False (Invalid) by a background process based on query expiration times. When V_T expires, the cache contents are replaced when there are table level changes in the database such as insertions and deletions. V_C is used for revalidation in case of updates to any of the columns in the table. Table 3 stores similar data for each column of each table in Table 2 - column names, expiry time intervals (validity period) for each column, and a column level validity flag (v_C). When v_C becomes False on expiry of any column in Table 3, the flag V_C is also set to False by the background process.

Table name (T)	Validity period (VP_T)	Table level validity flag (V_T)	Aggregate column validity flag (V_C)	Timestamp of last (re)validation (TS_T)
…				

Table 2 Proposed cache schema structure (Table level).

Table name (T)	Column name (C_C)	Validity period (VP_C)	Validity flag (v_C)	Timestamp of last (re)validation (TS_C)

Table 3 Proposed cache schema structure (Column level).

4.1.1 Cache Schema Management Algorithms

4.1.1.1 Schema Creation

1. Initialise schema in cache - store table/column names T and C_C from database schema in cache schema tables 2 and 3.
2. Set appropriate validity periods for tables and columns (VP_T, VP_C). (A table may become invalid upon row insertions/deletions, independent of any column-level updates.)
3. Set the three validity flags (V_T, V_C and v_C) to True (Valid*)* and set the timestamp columns (TS_T and TS_C) to the current date/time.

4.1.1.2 Schema Invalidation (executed by a background process)

```
// Loop through the table and column entries in Tables 2 and 3 at regular intervals.
for each table T in Table 2 do
        compare VP_T and TS_T
        if T has expired, set V_T = False        // Invalid
end do

for each table T in Table 3 do
    for each column C_C in T do
            compare VP_C and TS_C
            if C_C has expired then
                set v_C = False
                set V_c=False for corresponding T in Table 2
            end if
    end do
end do
```

4.2 Cache Query Management

We now describe how the web application server processes an input query using the cache and the cache schema and returns the query results. Our scheme considers caching of read (select) queries only - queries performing insert, delete and update operations are directly run on the backend database. We define a modified cache data structure as in Table 4 (compare Table 1) to store the input query entries. For each query, the names of the tables (T_Q) and columns (CS_Q) involved are also entered. The flag, Rx, indicates whether the query needs to be re-executed if the database contents have changed prior to revalidation.

Query (Q)	Table referenced in query (T_O)	Columnset referenced in query(CS_O)	Re-execute query flag (Rx)	Query Result (QR)
…				

Table 4 Proposed cache structure (Query results).

A typical database query involves the selection of one or more rows from one or more tables, each row containing one or more columns, to be displayed in a particular ordered fashion. A query may be contained in a URL or a web application written using JSP, ASP, CGI or any other script hosted on a web site. Each query result may be an HTML page parts of which are dynamically fetched from the database.

4.2.1 Cache Query Processing

When a query is invoked, the server inspects the schema entries for the referenced table and columns in Tables 2 and 3 to see if these objects have expired (Invalid). If the flags V_T and V_C are both True, then the cached query results are returned. If V_T = False, then the query has expired, and the database is checked for any changes at the table level (inserts or deletes). If there are changes, then the query is re-executed, and the results are cached and returned. All other queries in Table 4 which refer to the same database table/columns are then marked for re-execution, indicating that when these queries are invoked again by the application, they will be automatically re-executed without an additional visit to the database to check for changes.

If V_T in Table 3 is True, but the column level flag V_C is False, it means that one or more of the columns of the table in Table 2 have expired. If it is a column in the input query, this column is checked in the database for changes, and the query revalidated or rerun. All the other queries in Table 3 which also refer to the same database columns are then marked for re-execution, similar to above. This technique of automatic re-execution without having to revalidate against the database is found to reduce the number of effective visits to the database. Future discussions will refer to this technique as Schema-based Caching (Algorithm B).

Algorithm B: Schema-based Caching. Process query using a cached schema and return result.

Input: Query *Q*, Cache Tables 2-4.

Output: Query Result *QR*.

Method: //When a query is received, it is first checked to see if it already exists in the cache.

```
if Q ∈ Table 4 then     // query exists in Table 4
        if Rx = True for Q in Table 4 then
                if V_T = False or V_C = False for any of Q's tableset in Table 2 then
                        // Either Table-level or Column-level Flag is Invalid
                        execute Q
                        store new result into QR
                        set Rx = False
                        if V_T = False then
                                set V_T = True
                                reset TS_T to current time
                        end if
                        // Reset timestamps of this query's columns and mark the columns
                        // as Valid in Table 3
                        for all C_C ∈ CS_Q for query Q in Table 4 do
                                set v_c = True
                                reset TS_C
                        end do
                end if
                return QR
        else    // Rx = False
                // revalidate query
```

```
if V_T = True then      // Table level flag in Table 2 is valid
//means no row additions or deletions but column updates may have occurred
// check column valid flag of Table A
        if V_C = True then      //  Aggregate column validity  flag in Table 2
                                // is valid
                return QR
        else
                // Aggregate Column Validity flag of Table 2 V_C is Invalid.
                // Means that one  or  more columns in Table 3 are invalid.
                // Take  action if the column is part of the query Q. For each
                // of the columns  marked Invalid in query's columnset
                // take action.
                for each column C_C in Table 3 where v_c = False and
                        C_C ∈ CS_Q for Q in Table 4
                do
                        if C_C has been updated in database then
                                // need to re-query
                                set Rx= True
                                for all other queries in Table 4 that
                                    depend on C_C and whose Rx = False
                                        set Rx = True
                        end if
                end do
                if other than columns in this CS_Q no other column is invalid
                then
                        set V_C = True
                        reset TS_T
                end if
                if Rx = True for query = Q in Table 4 then //column changed
                        execute Q
                        store new result into QR
                        set Rx = False
                        return QR
                else
                        //columns expired and marked False (invalid)
                        // have not changed – detected on revalidation,
                        // no need to re-execute
                        return QR
                end if
                // Mark Valid and reset timestamps of query's columnset
                // in Table 3
                for each C_C ∈ CS_Q do
                        set v_c = True
                        reset TS_C
                end do
        end if
else
    //'Table valid' Column V_T in Table 2 is 'Invalid'
    // rows or columns may have been deleted or added
        if table T_Q in Q is updated in database then
                execute Q
                store new result into QR
                for all queries in Table 4 that depend on T_Q and whose
```

```
                                    Rx = False do
                                        set Rx = True
                                end do
                                return QR
                        end if
                        // Mark valid the relevant flags as entire table has changed and reloaded
                        for all T_Q in Tables 2 and 3 do
                                set V_T = True, V_C = True, v_C = True
                                reset TS_T, TS_C      // Reset timestamps for this table
                        end do
                end if
        end if
else // new query
        add Q to Table 4
        execute Q
        store new result into QR
        add Q's tables and columns to Table 2 and 3 from database schema, if not already cached
        set  V_T = True, V_C = True, v_c = True
        set TS_C, TS_T to current timestamp
        return QR
end if
```

5 Comparison of the Algorithms

We illustrate the working of our proposed web-caching scheme with a practical example and compare it with an existing standard web query caching scheme. For simplicity we use a relational database accessed with SQL queries, although our scheme is valid for any data source such as an XML database accessed by using XQuery or XPath[23].

As an example, consider a company's web site giving information about its products through a set of URLs/queries. The data is extracted from a relational database table called Products with the following structure:

Products(ProdID, Name, Type, Description, Price)

Most companies have a fixed product profile, and so all the columns in Products table except Price are not expected to vary much over time. Even when the price changes, it may be tolerable for some applications to provide a stale view of the existing price for some short period.

We will set a relatively shorter validity period of 1 day for the Price column compared to the other columns for which we use a longer interval of 90 days. We have a time frame of 200 days for Products table itself for revalidation.

Now consider the following web queries on the Products table:

Q1 -> Select * from Products order by ProdID;
Q2 -> Select Name, Type, Price from Products where ProdID='P12345';
Q3 -> Select Name, Description from Products where Type='Valve';
Let us illustrate both the Algorithms A and B with this example.

5.1 Algorithm A (Existing Scheme)

Table 5 shows the cache entries for this example with algorithm A in the format of Table 1. As noted above, Q1 and Q2 are set to expire after 1 day on account of the Price column, and Q3 is set to expire after a longer period of 90 days as its columnset does not include Price.

After day 1, Q1 and Q2 expire and are set to Invalid (Q_V = False) by the caching process. If Q1 is fired on, say, Day 3, and if Price has changed within that period, Q1 is re-executed and QR1 replaced. If Price has not changed, then Q1 is merely revalidated with a new timestamp and the existing QR1 is returned. If Q2 is now invoked on Day 5, it has expired as well and is processed for validation - database is checked and, depending upon whether Price has changed, Q2 is re-executed.

Let Q3 be invoked on Day 100, after it has expired and become Invalid. It too is checked for validation against the database. Since Q3 does not involve Price, there is no change in its results and Q3 is revalidated. The contents of Table 5 after processing queries are shown in Tables 6 and 7. The last two columns in these tables, shown with a *, are not part of cache structure but are included to indicate whether the database is checked for revalidation and, if the query-related data has changed in the database, whether the data is re-fetched from the database. Tables 6 and 7 depict the cases without and with the data change, respectively. The re-fetched data from the database replaces the existing cached query result.

Query	Validity Period (Days)	Query Validity Flag (Q_V)	Timestamp of last validation	Query Result
Q1	1	True	day 1	QR1
Q2	1	True	day 1	QR2
Q3	90	True	day 1	QR3

Table 5 Cache table structure for example to use with Algorithm A.

Query / URL	Validity Period (Days)	Query Validity Flag (V_Q)	Timestamp of last validation	Query Result	Database Checked*	Database Fetched*
Q1	1	False	day 3	QR1	Y	N
Q2	1	False	day 5	QR2	Y	N
Q3	90	False	day 100	QR3	Y	N

Table 6 Cache entries after processing of queries – when query-related data has not changed in database.

Query / URL	Validity Period (Days)	Query Validity Flag (V_Q)	Timestamp of last validation	Query Result	Database Checked*	Database Fetched*
Q1	1	False	day 3	QR1	Y	Y
Q2	1	False	day 5	QR2	Y	Y
Q3	90	False	day 100	QR3	Y	N

Table 7 Cache entries after processing of queries – when query-related data has changed in database.

5.2 Algorithm B (Proposed Scheme)

The initialized data structures for the example queries are shown in Tables 8-10.

In this case the expiry periods are set at the table and column levels, instead of setting them at the query level. If Price expires after 1 day, v_C for Price column in Table 9 is set to False. Flag V_C in Table 8 is also set to False (Invalid). If Q1 is fired afterwards, and Price has changed meanwhile, Q1 is re-executed and the results re-fetched into QR1. Hence database checked=Y, and database fetched=Y. This re-execution action is also propagated to Q2 in Table 10 since Q2 involves Price as well. Q2's Re-execute query flag is thus set. (Note: Only the flag is set, Q2 is not re-executed unless it is invoked.). When query Q3 is invoked, it, however, does not involve Price, and hence all its columns are valid in Table 9. Hence Q3 is answered with existing QR3 cached result. Tables 11 and 12 show the Table 10 cache entries after processing of the above queries by Algorithm B. Again the last two columns are not part of the cache structure but included for clarity.

A comparison of performance of algorithms A and B in terms of database visits during a single query validation is shown in Table 13. We note from the last column that algorithm B has better success in reducing database access. The main factor behind this saving is the fact that the proposed method provides granularity in setting validity periods for a query down to its column level rather than

stop at the query or table level. Therefore queries not containing a column are not affected by updates to that column and the cached results for such queries could be reused without revalidating against the database. The second factor is the propagation of a data change to other cached queries in Algorithm B enabling queries to be re-executed right away without performing additional checks for consistency. Moreover, since expiry periods are usually set with reference to the contents of the data, when they are set at the query level they will have to be set for each query executed against the same table. This may lead to inconsistent expiry settings across a set of queries accessing the same table. In our proposed scheme however, since expiry periods are set at the table and column levels they are set just once when the schema information is loaded into the cache.

Table name (T_T)	Validity period (VP_T)	Table Level Validity flag (V_T)	Aggregate Column Validity flag (V_C)	Time stamp of last (re)validation (TS_T)
Products	200	True	True	day 1

Table 8 Table level cache schema entry for the example queries.

Table name (T_T)	Column name (C_C)	Validity period (VP_C)	Validity flag (v_C)	Timestamp of last (re)validation (TS_C)
Products	Name	90	True	day 1
Products	Type	90	True	day 1
Products	Description	90	True	day 1
Products	Price	1	True	day 1

Table 9 Column level cache schema entries for the example queries.

Query	Table referenced in query	Columnsets referenced in query	Re-execute query flag	Query Results
Q1	Products	ProdId, Name, Type, Description, Price	False	QR1
Q2	Products	Name, Type, Price	False	QR2
Q3	Products	Name, Description	False	QR3

Table 10 Initial cache entries for the example queries.

Query	Table ref. in query	Columnsets referenced in query	Re-execute query flag	Query Results	Database Checked*	Database Fetched*
Q1	Products	ProdId, Name, Type, Description, Price	False	QR1	Y	N
Q2	Products	Name, Type, Price	False	QR2	Y	N
Q3	Products	Name, Description	False	QR3	N	N

Table 11 Cache entries after processing of queries – with no data change in database.

Query	Table ref. in query	Columnsets referenced in query	Re-execute query flag	Query Results	Database Checked*	Database Fetched*
Q1	Products	ProdId, Name, Type, Description, Price	False	QR1	Y	Y
Q2	Products	Name, Type, Price	True	QR2	N	Y
Q3	Products	Name, Description	False	QR3	N	N

Table 12 Cache entries after processing of queries – when data has changed in database.

Database has changed?	Algorithm used	Number of times database is checked	Number of times data is fetched	Total number of times database is accessed
Yes	A	3	2	5
	B	1	2	3
No	A	3	0	3
	B	2	0	2

Table 13 Comparative performance of algorithms A and B in terms of database visits for query validation.

6 Simulation and Results

The algorithms were compared by implementing a simulation of the cache manager using Oracle PL/SQL language. This cache manager, a process run by the web application server, received queries from the server and returned the query results. The query applications were written in Java Server Pages (JSP) that used ODBC calls for database access. [24].

The hardware setup consisted of a desktop client with a dual core processor, 2GB memory running Windows XP. The web server was Apache Tomcat running on an Intel Xeon 2-way processor with 8 GB RAM, cache size 250K, running under Linux. The database was Oracle 10g running on AIX OS. The database server used was an IBM Xeon 4-way processor with 8 GB RAM, with storage of 1000 GB.

A decision whether to fetch a query result from the cache or from the database depends on several conditions under which the query executes. The following factors were considered in simulating the various conditions for comparing algorithms A and B:-

- whether the query contains and is affected by the expiry/invalidation of one or more particular columns
- whether the query or the underlying table/columns have expired, requiring revalidation
- whether query-related data has changed in the database, requiring replacement
- whether the query is to be directly re-executed (without revalidation) due to change of related data in the database.

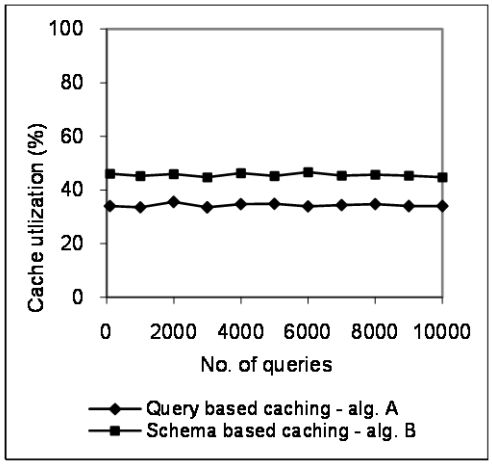

Figure 2 Cache utilization by algorithms A and B.

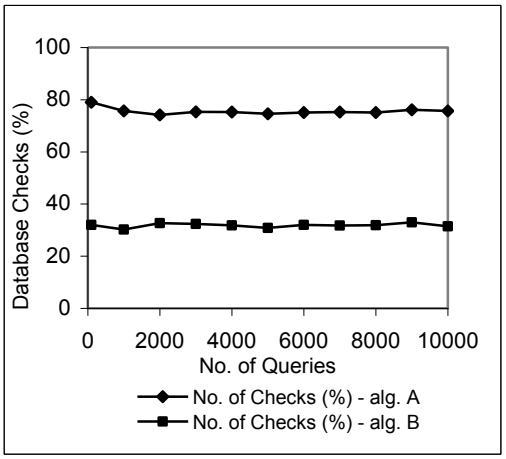

Figure 3 No. of database checks as percentage of no. of queries. Schema-based caching performs about 40% fewer checks on average than query-based caching.

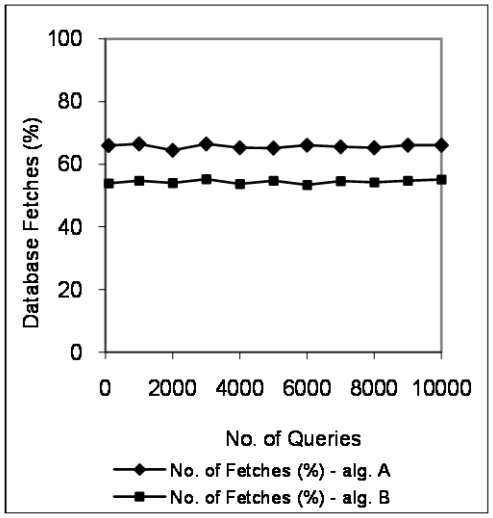

Figure 4 No. of database fetches as a percentage of no. of queries. Schema-based caching performs about 11% fewer fetches on average than query-based caching.

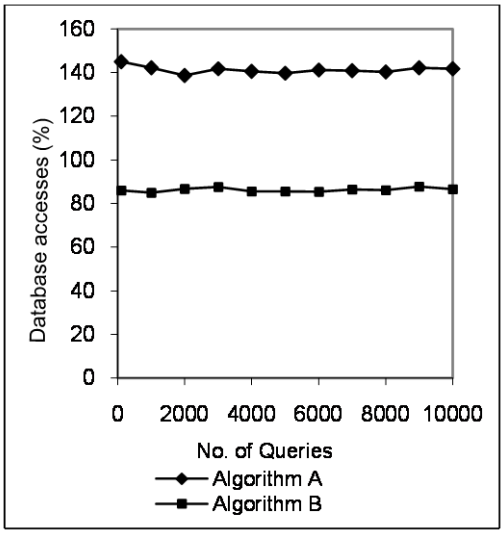

Figure 5 No. of database accesses as a percentage of no. of queries. Schema-based caching performs about 56% fewer accesses on average than query-based caching.

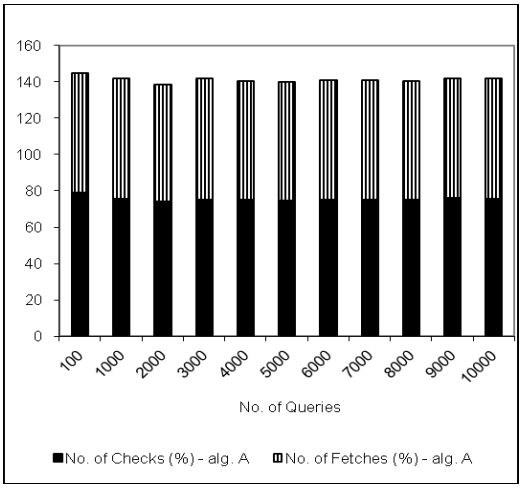

Figure 6 Database checks vs. fetches for query-based caching (algorithm A) as percentage of total no. of queries invoked

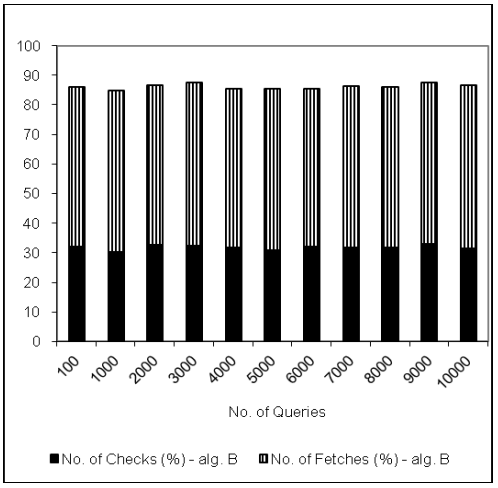

Figure 7 Database checks vs. fetches for schema-based caching (algorithm B) as percentage of total no. of queries invoked

Each simulation run consisted of passing to each algorithm identical query scenarios at identical random intervals. Each query scenario was a random sample from a population of all possible combinations of the above factors. The algorithm outputs were checked to see if and why the database was accessed and statistics were computed on the results using various criteria - number of database checks performed (for revalidation), number of database fetches carried out (for cache replacement), total number of database accesses made, etc.

Figure 2 shows the cache utilization of both the algorithms as a percentage of total number of queries. Query-based caching, on an average, uses the cache only 34% of the time whereas schema-based caching is more efficient by using it 45% of the time.

Figures 3 and 4 compare both the caching methods in terms of database checks and fetches as a percentage of total number of queries invoked during the simulation. The schema-based caching has fewer checks and fewer fetches than query-based caching. Figure 5 shows the overall comparison in terms of total number of database accesses made, where we note that schema-based caching performs about 56% fewer accesses than query-based caching.

Figures 6 and 7 summarize relative counts of checks and fetches by the two methods as a percentage of the total number of database accesses and provide another view into the simulation results in the form of bar charts. Note that a check need not necessarily be followed by a fetch, if for example the database has not changed, and that, in the case of algorithm B, a fetch can take place without a check. This enables algorithm B to be more efficient in terms of the ratio of checks to fetches. As the charts illustrate, not only does algorithm B make fewer visits to the database to check for cache consistency, it also follows up more of these visits by actual data fetches – i.e., a greater percentage of database visits are made only when necessary.

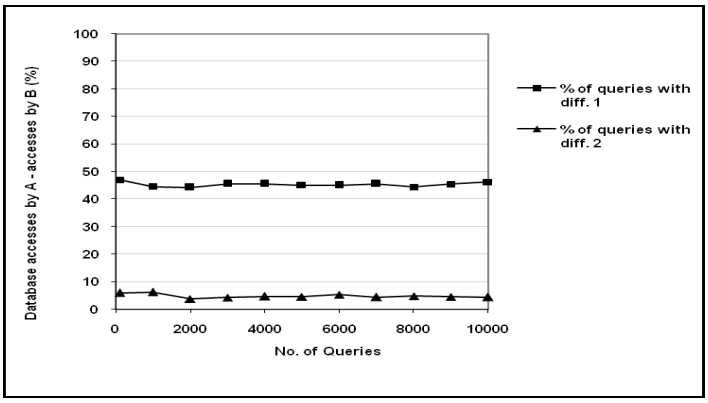

Figure 8 Difference in number of database visits by algorithms A and B as percentage of total number of queries.

We generated 64 possible test scenarios and used each one to give appropriate inputs to both the algorithms for comparison. Our analysis showed that in 32 of the cases, both the algorithms performed equally well – each resulting in same number of database visits. In 29 scenarios, B improved upon A in reducing the number of database accesses. In 3 cases there was a significant improvement in B's performance over A. These 3 cases pertained to scenarios such as one where a column had expired and its contents had changed in the database, but, since the query did not involve the column, the query result was not affected and therefore was returned from the cache itself. Therefore B accessed only the cache while A still accessed the database for revalidation checks and re-fetches. Figure 8 shows the simulation results in terms of the difference in number of database visits by A and B. When a check or fetch is made, it is counted as one visit. In 5% of the queries, algorithm A makes the maximum number of visits (both a check and a fetch i.e., 2 visits) to the database while algorithm B makes no visit at all (so the difference in number of visits is 2). In 46% of the cases, B makes 1 less visit than A (difference is 1). Thus we see the benefits when we reduce the query processing granularity to the schema column level, when database accesses are made only when necessary.

7 Conclusion

In this paper, we have proposed a method to improve query performance and QoS of web services by using an enhanced caching technique. The existing time-based solutions perform cache validation on an URL or application basis, and tend to access the database for every query, thus increasing the response time. The proposed method leverages the schema of the data source. It uses cache invalidation at table and column levels by duplicating the schema in the cache of the web application server. Thus database access is minimized for tables and columns that are common to several queries. In addition, those queries that do not access the expired columns do not visit the database at all for revalidation or re-fetch. Simulations were used to test the proposed method and the results show a significant reduction in database access. As the data source is usually an expensive resource shared by both web clients and non-web clients, reducing data access traffic by web queries will result in significant speed gains and quick response over the entire enterprise.

The proposed caching scheme could be applied to any type of data source such as an RDBMS or XML database, by copying the schema to the cache for query processing. It could also be used for both internet and intranet applications.

Acknowledgement

The authors are thankful to Dr. Soundar Rajan for his valuable contributions to this paper.

References

1. Adams, H. Best Practices for Web services: Web services performance considerations, Parts 1 &2. From http://www.ibm.com/developerworks/library/ws-best9/ and http://www.ibm.com/developerworks/library/ws-best10/. Last accessed Aug. 28, 2009.
2. Altinel, M., Luo, Q., Krishnamurthy, S., Mohan, C., Pirahesh, H., Lindsay, B.G., Woo, H. and Brown, L. 2002. DBCache: Database caching for Web Application servers. Proc. ACM SIGMOD 2002, 612.
3. Amiri, K., Park, S., and Tewari, R. 2002. A self-managing data cache for edge-of-network web applications. ACM CIKM 2002, 177-185.
4. Amiri, K., Park, S., Tewari, R. and Padmanabhan, S. 2003. DBProxy: A dynamic data cache for web applications. IEEE ICDE March 2003. From https://eprints.kfupm.edu.sa/33527/1/33527.pdf. Last accessed Oct. 20, 2009.
5. Amza, C., Soundararajan, G., and Cecchet, E. 2005. Transparent caching with strong consistency in dynamic content web sites. ACM ICS 2005, NY, 264-273. DOI = http://doi.acm.org/10.1145/1088149.1088185
6. Anbazhagan, M. and Nagarajan, A. Understanding quality of service for Web services. From http://www.ibm.com/developerworks/Webservices/library/ws-quality.html. Last accessed Aug. 28, 2009.
7. Apache HTTP Server version 2.2. From http://httpd.apache.org/docs/2.2/caching.html. Last accessed Aug. 28, 2009.
8. Candan, K.S., Li, W-S., Luo, Q., Hsiung, W-P. and Agrawal, D. 2001. Enabling dynamic content caching for database-driven web sites.ACM SIGMOD 2001.
9. Challenger, J., Iyengar, A. and Dantzig, P. A scalable system for consistently caching dynamic web data. 1999. Proc. IEEE INFOCOMM 99. https://eprints.kfupm.edu.sa/20773/1/20773.pdf. Last accessed Oct. 20, 2009.
10. Chatterjee, S. and Webber, J. Developing Enterprise Web Services. Pearson Education, 2004.
11. Chidlovskii, B. and Borghoff, U. 2000. Semantic caching of Web queries. The VLDB Journal 9, 1 (2000), 2-17. DOI = http://dx.doi.org/10.1007/s007780050080
12. Degenaro, L., Iyengar, A., Lipkind, I., and Rouvellou, I. 2000. A middleware system which intelligently caches query results. IFIP/ACM Intl. Conf. Distributed Systems Platforms, 24-44. From http://www.research.ibm.com/AEM/documents/abr_mw2000.pdf. Last accessed Oct. 20,

2009.
13. Datta, A., Datta, K., Thomas, H.M., VanderMeer, D.E., Ramamritham, K., and Fishman, D. 2001. A Comparative study of alternative middle tier caching solutions to support dynamic web content acceleration. Proc. 27th VLDB 2001, Rome, Italy. From http://www.cse.iitb.ac.in/~krithi/papers/vldbj_chutney.pdf. Last accessed Oct. 20, 2009.
14. Datta, A., Datta, K., Thomas, H.M., VanderMeer, D.E., Suresha and Ramamritham, K. 2002. Proxy-based acceleration of dynamically generated content on the world wide web: an approach and implementation. Proc. ACM SIGMOD, June 2002. From http://dsl.serc.iisc.ernet.in/pub/chutney.pdf. Last accessed Oct. 20, 2009.
15. Galindo-Legaria, C., Grabs, T., Kleinerman, C., and Waas, F. 2005. Database change notifications: primitives for efficient database query result caching. In Proceedings of the 31st international Conference on Very Large Data Bases. VLDB Endowment, 1275-1278. From http://www.vldb2005.org/program/paper/demo/p1275-galindo-legaria.pdf. Last accessed Oct. 20, 2009.
16. HTTP. Hypertext Transfer Protocol -- HTTP/1.1. From http://www.w3.org/Protocols/rfc2616/rfc2616.html. Last accessed Aug. 28, 2009.
17. Jasnowski, N. Java, XML & Web Services Bible. IDG Books India, 2002.
18. Kossmann, D., Franklin, M. J., Drasch, G., and Ag, W. 2000. Cache investment: integrating query optimization and distributed data placement. ACM Trans. Database Syst. 25, 4 (2000), 517-558. From http://www.dbis.ethz.ch/research/publications/27.pdf. Last accessed Oct. 20, 2009.
19. Larson, P.A., Goldstein, J., Guo, H. and Zhou, J. 2004. MTCache: Transparent Mid-tier Database Caching in SQL Server. Proc. IEEE Intl. Conf. Data Engg. 2004, 177-189.
20. Luo, Q., Krishnamurthy, S., Mohan, C., Pirahesh, H., Woo, H., Lindsay, B.G., and Naughton, J. F. 2002. Middle-tier Database caching for e-Business. ACM SIGMOD June 2002. From http://www.cse.ust.hk/catalac/papers/dbcache_sigmod02.pdf. Last accessed Oct. 20, 2009.
21. Luo, Q., Naughton, J. F., Krishnamurthy, R., Cao, P., Li, Y. 1999. Active Query Caching for Database Servers. Webdatabase 2000: 29-34. From https://eprints.kfupm.edu.sa/22643/1/22643.pdf. Last accessed Oct. 20, 2009.
22. Luo, Q., Naughton, J. F., and Xue, W. 2008. Form-based proxy caching for database-backed web sites: keywords and functions. The VLDB Journal 17, 3 (May. 2008), 489-513. DOI= http://dx.doi.org/10.1007/s00778-006-0018-x
23. Mandhani, B. and Suciu, D. 2005. Query caching and view selection for XML databases. VLDB 2005, 469-480. From http://www.vldb2005.org/program/paper/wed/p469-mandhani.pdf. Last accessed Oct. 20, 2009.
24. Mogha , R. and Preetham, Java Web Services Programming. Wiley Dreamtech, 2003.
25. Mohan, C. 2001. Tutorial: Caching technologies for web Applications. VLDB'01, From http://www.almaden.ibm.com/u/mohan/Caching_VLDB2001.pdf. Last accessed Aug. 28, 2009.
26. Nottingham, M. Caching Tutorial for Web Authors and Webmasters. From http://www.web-caching.com/mnot_tutorial/. Last accessed Oct. 20, 2009.
27. Oracle Corporation. 2005. Oracle Application Server Web Cache 10*g*. September 2005. From http://www.oracle.com/technology/products/ias/Web_cache/pdf/WebCache1012_twp.pdf. Last accessed Aug. 28, 2009.
28. Turaga, R., Cline, O. and Van Sickel, P. WebSphere Application Server Step by Step. MC Press, 2006.
29. Tolia, N. and Satyanarayanan, M. 2007. Consistency-preserving caching of dynamic database content. ACM WWW 2007, NY, 311-320. From http://www2007.org/papers/paper120.pdf. Last accessed Oct. 20, 2009.
30. Yagoub, K., Florescu, D., Issarny, V., Valduriez, P. 2000. Caching strategies for data-intensive web sites. Proc. VLDB, 188-199. From http://www-roc.inria.fr/arles/doc/ps00/vldb.pdf. Last accessed Oct. 20, 2009.
31. Zhang, L., Floyd, S. and Jacobson, V. Adaptive Web caching. In NLANR Web Cache Workshop 97. From http://citeseer.ist.psu.edu/zhang97adaptive.html. Last accessed Oct. 20, 2009.