Accelerating Database Processing at e-Commerce Sites

Seunglak Choi, Jinwon Lee, Su Myeon Kim, Junehwa Song, and Yoon-Joon Lee

Korea Advanced Institute of Science and Technology
373-1 Kusong-dong Yusong-gu Daejeon 305-701, Korea

Abstract. Most e-commerce Web sites dynamically generate their contents through a three-tier server architecture composed of a Web server, an application server, and a database server. In such an architecture, the database server easily becomes a bottleneck to the overall performance. In this paper, we propose WDBAccel, a high-performance database server accelerator that significantly improves the throughput of the database processing, and thus that of the overall Web site. WDBAccel eliminates costly, complex query processing needed to obtain query results by reusing previous query results for subsequent queries. This differentiates WDBAccel from other database cache systems, which replicate a database into multiple conventional DBMS's and distribute queries among them. We evaluate the performance of WDBAccel by using the queries of the TPC-W benchmark. The measurement results show that WDBAccel outperforms DBMS-based cache systems by up to an order of magnitude.

1 Introduction

With the explosive growth of the Internet, numerous value-generating services are provided through WWW. In most e-commerce Web sites, those services are usually deployed on a three-tier server architecture, which consists of Web servers, application servers, and database servers. Web contents are dynamically generated upon a request through such system components. Due to its low scalability and complexity of query processing, a database server is a major bottleneck to the overall site performance.

We propose a high-performance database server accelerator, called WDBAccel, which significantly improves the throughput of database processing in multi-tier Web sites (see Figure 1). The main approach of WDBAccel is to cache results of frequently-issued queries and reuse these results for incoming queries. Upon a query hit, the query result is immediately served from the cache. WDBAccel keeps cached results consistent to an origin database by invalidating staled results. In addition, WDBAccel is designed to use main memory as the primary storage, minimizing disk operations. Thus, WDBAccel can improve the performance of database processing more than an order of magnitude.

WDBAccel differs from other existing DB cache system [6, 11, 9, 1] in that the primary purpose of WDBAccel is to accelerate the database processing. On

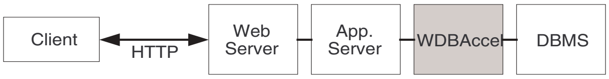


Fig. 1. WDBAccel deployment in an e-commerce Web site

the contrary, the primary purpose of other cache systems is to distribute the load of database processing into multiple cache nodes. Those systems replicate a database into multiple nodes and distribute queries among them. In serving queries, they rely on the underlying DBMS, which executes costly query processing to obtain a query result. Thus, the performance of DBMS-based cache systems is limited by their underlying DBMS.

Our WDBAccel design incorporates three policies that effectively utilize the limited space of main memory. First, WDBAccel employs the *derived matching.* Even when the cache does not store the identical query result, the result for a query can be derived from one or more previously stored queries. In many Web-based applications, selection regions of queries tend to overlap each other. Thus, WDBAccel performs the derived matching by containment checking among selection regions of stored query results. Second, WDBAccel removes the storage redundancy of the tuples belonging to two or more query results. In many cases, query results contain identical tuples. Therefore, WDBAccel eliminates such a storage redundancy by storing query results in the unit of tuples. Third, a cache replacement policy in WDBAccel evaluates storage cost in a different way from other Web caching systems. In many systems, the policy considers the size of cached data items. However, it is not appropriate in WDBAccel due to the tuples shared among multiple query results. WDBAccel considers both the size of query results and shared tuples.

WDBAccel provides several advantages to database-driven Web sites. The most competitive advantage is that it drastically improves the throughput of the Web sites. Second, WDBAccel reduces the total cost of ownership. WDBAccel is a light-weight system optimized in caching and serving query results. Thus, it can be deployed even on lower-end H/W system while achieving a high level of performance. Third, WDBAccel can be easily deployed as a middle-tier solution between the Web application server and the database server. By supporting the standard interfaces like JDBC or ODBC, WDBAccel does not require Web applications to be modified. Forth, the high-performance nature of WDBAccel reduces the total number of cache nodes managed by an administrator, reducing administration cost.

This paper is organized as follows. In section 2, we describe the architecture of WDBAccel. In section 3, we explain technical details including query matching, cache storage, and cache replacement. In section 4, we evaluate and analyze the performance of WDBAccel. In section 5, we describe related works and compare them to our system. Finally in section 6, we present conclusions.

2 System Architecture

WDBAccel is a Web database server accelerator which processes queries delivered from the front-side Web application servers (WAS's) on behalf of database servers. It is designed as a middle-tier system which can be deployed between WAS's and database servers without extensive modification to either system as other middle-tier cache systems [6, 11, 9]. To deploy WDBAccel on an existing e-commerce service system, it is only required to change an existing database driver to WDBAccel's driver at the WAS.

WDBAccel is a query result caching system. As mentioned, it stores the results of previous queries and then serves incoming queries delivered from WAS. The query result caching is extremely useful when a workload is read-dominant. The workload of most e-commerce applications is read-dominant. In e-commerce sites, visitors spend the most time finding and reading some information, *e.g.*, product catalogs, news, articles, etc. Update interactions such as ordering products are relatively very infrequent. For example, the TPC-W benchmark [12][1] specifies that the portion of read queries, in an average case, is 80% of the entire workload. Thus, we can expect that the query result caching will show a high level of performance in many e-commerce sites.

Figure 2 shows the overall architecture of WDBAccel and the processing flow. The WAS sends a query to the Query Redirector (QR) (1), then QR checks whether the given query is *read* or *write*. If the query is a *write*, it sends the query to the Consistency Maintainer (CM) (A). CM forwards the query to the origin database server (B) and performs the process to maintain cache consistency. If the query is a *read*, then it forwards the query to the Fragment Processor (FP) (2). FP references the Cache Dictionary (CD) to decide whether the result for the incoming query can be constructed based on cached fragments (3). A fragment is a query result stored in a cache. If fragments for the incoming query are found, FP retrieves the fragments (4). Otherwise, FP sends the query to the database server (a) and receives the result for the query (b). Then, it forwards both the query and the query result to the Cache Controller (CC) (c). CC inserts the query into CD and the query result into the Cache Pool (CP) (d). (When the cache does not have enough space to store new query results, CC executes a cache replacement algorithm.) FP constructs and sends the query result to QR (5). Finally, QR sends the query result to the WAS (6).

CD is a collection of meta information about fragments stored in the Cache Pool (see Figure 3). Each entry stores meta information on a fragment (*e.g.*, selection region, fragment size, and pointer to data). Entries are classified into different *query groups* according to the structure of queries. Then, each group is indexed by selection regions of the queries. The index is used to reduce the search time for fragments matching a query.

The caching inherently incurs the inconsistency between cached results and an origin database. If a data element in an origin database is updated, the fragments derived from the updated data will be stale. WDBAccel includes CM which ensures the consistency. The primary goal of CM is to minimize the consistency overhead. It matches an update against the common parts of many

[1] The TPC-W benchmark is an industrial standard benchmark to evaluate the performance of database-driven Web sites. It models an e-commerce site (specifically, an online bookstore).

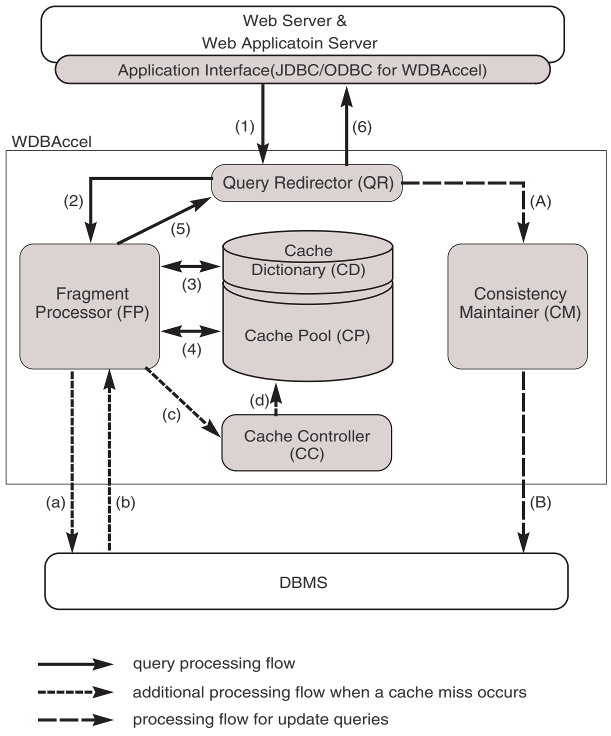


Fig. 2. WDBAccel architecture and processing flows

queries, not individual queries. Thus, it avoids repeated matching of each query. In addition, CM ensures strong cache consistency by invalidating affected query results before the completion, *i.e.*, the transaction commit, of a database update. For some pages, strong consistency is critical to service them always in an up-to-date version (*e.g.* the cost of products in Web shopping sites). [5] describes our consistency mechanism in detail.

3 Technical Details

3.1 Derived Matching

WDBAccel employs the derived matching to maximize the main-memory hit ratio, *i.e.*, the rate of reusing query results cached in main memory. When WDBAccel fails to find an exactly matching fragment, it tries to find one or more fragments from which a query result can be derived. We give an example of the derived matching. In this example, a query Q can be derived from the union of fragments F_1 and F_2 although Q does not exactly match either F_1 or F_2.

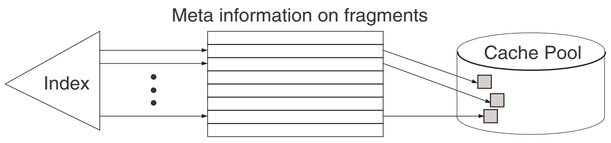


Fig. 3. Cache Dictionary

Example 1. Given fragments and a query as follows,

F_1 : `SELECT * FROM ITEM`
`WHERE I_PUB_DATE >= '01/01/2003' AND I_PUB_DATE <= '01/20/2003'`
F_2 : `SELECT * FROM ITEM`
`WHERE I_PUB_DATE >= '01/10/2003' AND I_PUB_DATE <= '01/30/2003'`
Q : `SELECT * FROM ITEM`
`WHERE I_PUB_DATE >= '01/05/2003' AND I_PUB_DATE <= '01/25/2003'`

Q can be derived from the union of F_1 and F_2 since the selection region of the union contains the selection region of Q.

WDBAccel utilizes the *selection region dependency* in order to maximize finding the derived matching. A selection region dependency is a computational dependency in selection regions. In example 1, a query Q has the selection region dependency on a union of a fragment F_1 and a fragment F_2 in that the selection region of the union contains the selection region of Q. By investigating the dependencies among selection regions, WDBAccel is highly likely to find a derived matching. This is due to the characteristic of the queries used in many Web-based applications: selection regions from different queries with the same template tend to overlap each other and may form a hot range. In example 1, selection regions on `I_PUB_DATE` will frequently fall together near to the present time. This is because customers in an online bookstore prefer to select new books.

When a query is matched by a derived matching, the process of the *query result deriving* follows the matching process. For the query result deriving, two operations, *union* and *trim*, are applied to matching fragments. The union operation is used to merge the matching fragments and to eliminate tuple duplication. In the Example 1, by the union, the tuples of the matching fragments F_1 and F_2 are merged and the duplication of the tuples ('01/10/2003' $\leq$ `I_PUB_DATE` $\leq$ '01/20/2003') is removed. We used the selection attributes of tuples to identify duplicate tuples. They are located in overlapping selection regions among the matching fragments.

The trim operation is used to cut off the tuples that do not constitute a query result when matching fragments contain a query. In the Example 1, the tuples ('01/01/2003' $\leq$ `I_PUB_DATE` $<$ '01/05/2003') of the matching fragment F_1 and the tuples ('01/25/2003' $<$ `I_PUB_DATE` $\leq$ '01/30/2003') of the matching fragment F_2 is cut off by the trim operation. In order to determine whether a tuple is a part of a query result, the attribute values of the tuple is compared to the selection predicates of the query.

3.2 Cache Storage

To increase the utilization of a limited cache storage, it is crucial to avoid redundant storage of identical data. Query results can include identical tuples. In example 1, the tuples located in the overlapping selection region (`I_PUB_DATE`, ‘01/10/2003’, ‘01/20/2003’) can be included in both fragments F_1 and F_2. As mentioned in section 3.1, overlaps among query results as above are common in many Web applications.

To remove the redundant storage, WDBAccel identifies overlaps in selection regions of fragments and eliminates redundant storage of tuples. The storage policy of WDBAccel stores query results in the unit of tuples. Before storing each tuple in a new query result, it determines if the tuple already exists in the cache. This is done by comparing the values of the selection attributes of each tuple with those of the cached tuples. To speed up the comparison, the policy scans only the tuples of the fragments overlapping the new query result. Note that these fragments have already been retrieved in the query matching process. We refer to the Cache Pool adopting this policy as the *Cache Pool in the unit of Tuples* (CPT).

3.3 Cache Replacement

Hit ratio is improved if the cache stores the query results which are frequently accessed and consume less storage space. Thus, we evaluate the profit of a query result as follows:

$$profit(f) = \frac{popularity(f)}{s\_cost(f)}$$

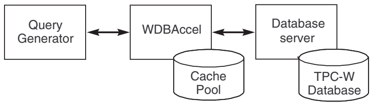
where $s\_cost(f)$ is the storage cost of a fragment f and $popularity(f)$ represents the popularity of a fragment f. When a cache space is full, the Cache Controller evicts the query result with the lowest $profit$ value (called a victim). Usually, the last access time or the number of accesses are used for *popularity*. Under CPT, we consider that some tuples are shared among multiple fragments. We divide the storage cost of a shared tuple among sharing fragments. In this case, *s_cost* is computed as follows.

$$s\_cost(f) = \sum_{t_i \in T(f)} size(t_i)/n\_frag(t_i)$$

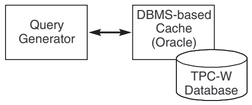
where $T(f)$ is a set of tuples belonging to a fragment f, $size(t_i)$ is the size of a tuple t_i, and $n\_frag(t_i)$ is the number of the fragments containing a tuple t_i.

4 Experiments

In this section, we compare the processing capability of WDBAccel with that of DBMS-based systems by measuring their throughputs.



(a) WDBAccel



(b) DBMS-based cache system

Fig. 4. Experimental setup

Experimental Setup. Figure 4 (a) shows the setup for evaluating the WDBAccel system. The *Query Generator* emulates a group of Web application servers, generating queries. It runs on a machine with a Pentium III 1GHz, 512MB RAM. We implemented the prototype of WDBAccel which included all components and core functions described above. WDBAccel is deployed between the Query Generator and the database server. WDBAccel runs on the machine with a Pentium III 1GHz, 512M RAM. For the origin database server, we used Oracle 8i with the default buffer pool size of 16MB. The database server runs on a machine with a Pentium IV 1.5GHz, 512M RAM. We populated the TPC-W database in the database server at two scales: 10K and 100K (cardinality of the `ITEM` table)[2]. All three machines run Linux and are connected through a 100Mbps Ethernet.

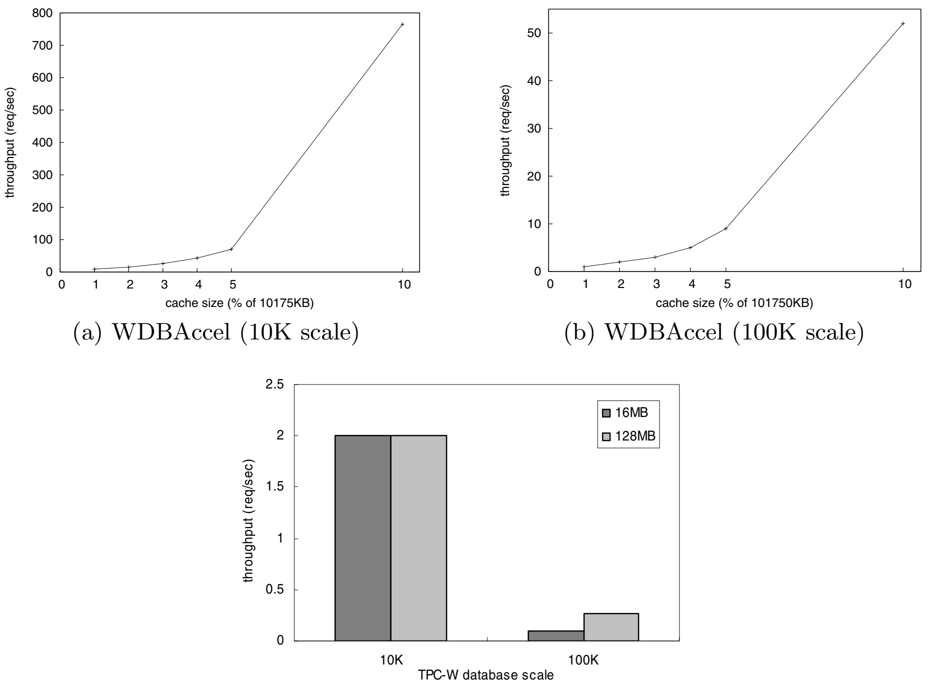
Figure 4 (b) shows the experimental setup for evaluating the throughput of the DBMS-based cache system. The DBMS-based system conceptually consists of the cache-related components and the underlying DBMS. We used the simplified system by omitting the cache-related components. For the underlying DBMS, we used Oracle 8i. We also assume that the DBMS-based system achieves 100% cache hit ratio. Thus, the Oracle database is fully populated with the entire TPC-W database. Note that the throughput measured under this simplified setup will be higher than that taken under a real situation using the DBMS-based cache systems. Both systems run on the Linux machine with a Pentium III 1GHz, 512MB RAM and are connected through a 100Mbps Ethernet.

Workload Traces. We used for experiments the trace of the search-by-title query specified in TPC-W. This query searches for all the books whose titles include the keyword specified by users. It is the most frequently-used query in TPC-W; in an average case, its usage frequency constitutes 20% of the entire TPC-W workload. We believe that the query is frequently used in many e-commerce sites in general. The following is the query template of the query. We refer to the search-by-title query as the *keyword query.*

```
SELECT TOP 50 I_TITLE, I_ID, A_FNAME, A_LNAME FROM ITEM, AUTHOR
WHERE I_A_ID = A_ID AND I_TITLE LIKE ‘%@Title%’ ORDER BY I_TITLE
```

Performance Comparison Between WDBAccel and the DBMS-Based Cache Systems. The experiment was performed through two steps: the *fill phase* and the *test phase*. In the fill phase, we fill the cache space with fragments

[2] In TPC-W, the scale of database is determined by the cardinality of ITEM table.



(a) WDBAccel (10K scale) (b) WDBAccel (100K scale)

(c) DBMS-based cache system

Fig. 5. The throughputs for the keyword query

by issuing 100,000 queries. This phase warms up the cache space so that the throughput can be measured under normal condition, *i.e.,* within a normal range of cache hit ratio, during the test phase. The cache size of WDBAccel is set to range from 1% to 10% of the sum of `ITEM` and `AUTHOR` table sizes. `ITEM` and `AUTHOR` are the tables which are accessed by the keyword and the range queries. The sums of two table sizes corresponding to the 10K and 100K scales of TPC-W are around 10MB and 100MB, respectively. The buffer pool size of the DBMS-based system is set to 16MB and 128MB.

Figure 5 shows the results for the keyword query. It shows that WDBAccel outperforms the DBMS-based system by an order of magnitude. This implies that reusing query results can significantly save the query processing cost. In ordinary database servers, the keyword query needs a linear search on the `ITEM` table. This is because the keyword query uses the LIKE operator, which includes wild card characters and does a pattern matching. Such a LIKE operator cannot benefit from the index structure on the attribute `I_TITLE`. On the other hand, in WDBAccel, the query processing time increases slightly with the number of fragments no matter what a query type is. Also, WDBAccel does not require any disk accesses.

Another interesting observation is that the throughput of WDBAccel rapidly improves as the cache size increases. With the cache size up to 5%, the origin database server is the bottleneck. In that case, the cache miss ratio (and thus hit ratio) is an important factor of the throughput. For example, if the cache miss ratio decreases from 2% to 1%, the throughput will be doubled.

5 Related Work

Recently, several query result caching systems have been reported in the context of dynamic Web content services. Form-based cache [10] is the first effort for a query result caching. It extended the URL-based proxy caching for active proxy query caching with limited query processing capability. The proposed framework could effectively work for a top-n conjunctive queries generated from the HTML forms. However, it only addressed keyword queries. Weave [13] caches data in the unit of XML and HTML pages as well as query results. Weave focuses on the declarative specification for Web sites through a logical model and a customizable cache system that employs a mix of different cache policies. DBProxy [1] employs a conventional DBMS for storing and retrieving query results like the DBMS-based caching systems. Therefore, its performance could be limited by the complex query processing of the underlying DBMS. The main focus of WDBAccel is to efficiently scale the performance of database-driven Web sites. It provides a framework for a high-performance query result caching and an efficient storage structure for it.

The *HTML caching* has also been used to improve the performance of e-commerce sites. It selects cacheable data in the unit of the whole HTML page or HTML components which are parts of a HTML page. It then caches the data in front of Web servers or inside WAS's. We call the former *HTML page caching* [8, 3] and the latter *HTML component caching* [4, 7, 2]. The main advantage of the HTML caching is that the performance gain on a cache hit is better than that of others. This is because it saves much or all of the cost of HTML page generation as well as database processing. However, the HTML page caching is not effective in caching dynamic pages. HTML component caching can be effective to some extent in caching dynamic contents. A problem in the component caching is that it incurs high administration cost; cache administrators should go through a complex process of marking cacheable and non-cacheable units.

6 Conclusions

We have presented the design and implementation of a high-performance database server accelerator. WDBAccel improves throughput of the database processing, which is a major bottleneck in serving dynamically generated Web pages. To improve the performance, it reuses previous query results for subsequent queries and utilizes main memory as a primary cache storage. WDBAccel performs the derived matching to effectively find a set of query results required to construct a result of an incoming query. It employs the storage policy that reduces storage redundancy. In addition, the cache replacement policy takes into account storage costs of query results and overlaps among them. The experimental results show that WDBAccel outperforms DBMS-based cache systems by up to an order of magnitude.

References

1. Khalil S. Amiri, Sanghyun Park, Renu Tewari, and Sriram Padmanabhan. DBProxy: A self-managing edge-of-network data cache. In *19th IEEE International Conference on Data Engineering*, 2003.
2. Jesse Anton, Lawrence Jacobs, Xiang Liu, Jordan Parker, Zheng Zeng, and Tie Zhong. Web caching for database applications with oracle web cache. In *Proceedings of ACM SIGMOD Conference*, 2002.
3. K. Selcuk Candan, Wen-Syan Li, Qiong Luo, Wang-Pin Hsiung, and Divyakant Agrawal. Enabling dynamic content caching for database-driven web sites. In *Proceedings of ACM SIGMOD Conference*, Santa Barbara, USA, 2001.
4. Jim Challenger, Arun Iyengar, Karen Witting, Cameron Ferstat, and Paul Reed. A publishing system for efficiently creating dynamic web content. In *Proceedings of IEEE INFOCOM*, 2000.
5. Seunglak Choi, Sekyung Huh, Su Myeon Kim, JuneHwa Song, and Yoon-Joon Lee. An efficient update management mechanism for query result caching at database-driven web sites. *Under submission.*
6. Oracle Corporation. Oracle9ias cache.
   http://www.oracle.com/ip/deploy/ias/index.html?cache.html.
7. Anindya Datta, Kaushik Dutta, Helen Thomas, Debra VanderMeer, Suresha, and Krithi Ramamritham. Proxy-based acceleration of dynamically generated content on the world wide web: An approach and implementation. In *Proceedings of ACM SIGMOD Conference*, 2002.
8. Vegard Holmedahl, Ben Smith, and Tao Yang. Cooperative caching of dynamic content on a distributed web server. In *Proceedings of the 7th IEEE International Symposium on High Performance Distributed Computing*, 1998.
9. Qiong Luo, Sailesh Krishnamurthy, C. Mohan, Hamid Pirahesh, Honguk Woo, Bruce G. Lindsay, and Jeffrey F. Naughton. Middle-tier database caching for e-business. In *Proceedings of ACM SIGMOD Conference*, 2002.
10. Qiong Luo and Jeffrey F. Naughton. Form-based proxy caching for database-backed web sites. In *Proceedings of the 27th VLDB Conference*, Roma, Italy, 2001.
11. TimesTen Performance Software. Timesten library.
    http://www.timesten.com/library/index.html.
12. Transaction Processing Performance Council (TPC). TPC benchmark$^{\text{TM}}$W (web commerce) specification version 1.4. February 7, 2001.
13. Khaled Yagoub, Daniela Florescu, Valerie Issarny, and Patrick Valduriez. Caching strategies for data-intensive web sites. In *Proceedings of the 26th VLDB Conference*, 2000.