Database Server Workload Characterization in an E-commerce Environment[*]

Fujian Liu Yanping Zhao Wenguang Wang Dwight Makaroff

Department of Computer Science
University of Saskatchewan
57 Campus Drive
Saskatoon, SK, S7N 5A9, Canada
{ful113, zhao, wang, makaroff}@cs.usask.ca

Abstract

In an E-commerce system, the database server performance is crucial. Dynamic cache is often used to reduce the load on the database server, which reduces the need for scalability. A good understanding of the workload characteristics of the database server in an E-commerce environment is important to the design, tuning, and capacity planning of the database server. In this paper, we characterize the database server workloads in a benchmark E-commerce system. We focus on the response time, CPU utilization, the database page reference characteristics, and disk I/Os of the database server. We found that using dynamic cache can substantially reduce the CPU utilization but not always the number of disk I/Os of the database server. In most cases, using dynamic cache reduces temporal locality in database page references, but to a smaller degree than that reported in file servers and web proxies. Interestingly, in certain E-commerce workloads, using dynamic cache results in better temporal locality.

1. Introduction

The use of the Internet to conduct E-commerce activities has shown a steady increase and is a major component of the retail industry. A typical corporation-sized E-commerce system is composed of many front-end servers and a back-end database server. The front-end servers include web/application servers, image servers, and dynamic cache servers. They interact with users through web interface and execute business logic. The back-end database server stores business information and processes queries.

In an E-commerce system, the front-end servers can be replicated easily because they store read-only data (e.g., images, scripts, and static web pages). It is cost-effective to use low-end commodity machines for this purpose. Alternatively, the database server stores frequently updated data. It is difficult to replicate the database server while maintaining data consistency efficiently. A single large database server is often used in such systems. This database server may easily become the performance bottleneck, since most user requests invoke database queries. Also, the database server is often the most expensive component in the system. For example, in the xSeries 440 system [16], which generates the highest TPC-W benchmark performance result, the price of the database server is about the same as that of the sixty-two front-end servers. Therefore, improving the performance of the database server is crucial to improve the overall performance of the system and to reduce the cost.

Understanding the workload characteristics of the database server is a prerequisite for studying its performance. The DBMS is designed and tuned for traditional database workloads, e.g., On-line Transaction Processing (OLTP) and On-line Analytical Processing (OLAP). E-commerce workloads may have different characteristics. Moreover, *dynamic cache* is often employed to reduce the load on the database server. Various dynamic cache technologies may be used in the system, and new caching approaches may emerge in the future. The existence of dynamic cache may affect the workload characteristics of the database server.

In this paper, we study the workload characteristics of the database server in a large E-commerce system, and investigate the impact of dynamic cache on the workload characteristics. The web server(s), the application server(s), and the database server in the system that we emulated reside on separate machines. We focus on the response time,

* This research is supported by NSERC and IBM Centres for Advanced Studies.

CPU utilization, the database page reference characteristics, and disk I/Os of the database server.

We use the TPC-W benchmark [16] in this study. We emulated three kinds of dynamic caches, *query result cache*, *table cache*, and a hybrid of both termed as *hybrid-cache*. A query result cache stores query results and serves subsequent identical queries using these results. A table cache server runs on a separate machine and replicates infrequently-updated tables. The table cache server processes queries that involve the replicated tables so as to reduce the load on the database server. The hybrid-cache combines query result cache and table cache to achieve better caching efficiency. We implemented a light-weight trace tool in the DBMS and collected traces of database page references to the buffer pool (an in-memory buffer that caches database pages). We analyzed trace characteristics, and evaluated various buffer replacement algorithms using trace-driven simulations.

The main findings of this study are:

- Using dynamic cache can considerably reduce the CPU utilization of the database server.
- Query result cache has little impact on the number of disk I/Os of the database server, while table cache and hybrid-cache can significantly reduce the number of disk I/Os.
- Table cache and hybrid-cache can substantially reduce the working set size of the database page references.
- In most cases, using dynamic cache reduces temporal locality in database page references, but to a smaller degree than that reported in file servers [17] and web proxies [6]. In workloads with few cacheable queries, however, using table cache increases temporal locality.
- Various buffer replacement algorithms work well in systems using dynamic cache. LFU performs better than LRU in systems using dynamic cache. Advanced replacement algorithms based on temporal locality, such as LIRS and LRU-2, perform the best in both systems with and without dynamic cache.

Although we use a benchmark E-commerce system rather than a real system, the analysis methodology employed in this paper is general. We use the benchmark system because it is very hard to collect database server traces in real E-commerce systems. The benchmark system provides a controlled environment to emulate most key features that affect the E-commerce system performance.

The remainder of this paper is organized as follows. We first provide the background and then describe the methodology used. After that, we present the workload characteristics at the database server and discuss the results. Finally we conclude the paper and outline possible future research.

2. Background

2.1. The TPC-W benchmark

The TPC-W benchmark [16] is a widely used benchmark for measuring the performance of E-commerce systems. It simulates a breadth of activities of a business-oriented transactional web server, specifically an on-line bookstore. The store size is expressed by the number of items and the number of emulated browsers (EBs). The performance metric reported by TPC-W is the number of web interactions processed per second (WIPS).

TPC-W defines two classes of web interactions: *browse* and *order*. The *browse* web interactions involve browsing the web site and searching the database, e.g., querying new products, best sellers, and product details. The *order* web interactions update the database, e.g., loading shopping carts, and registering customers. By varying the ratio of the *browse* and *order* web interactions, TPC-W simulates three kinds of workloads: *shopping*, *browsing*, and *ordering*. Table 1 describes the distribution of interactions in the workloads.

Workload Type	*Browse* Interactions	*Order* Interactions
Browsing	95%	5%
Shopping	80%	20%
Ordering	50%	50%

Table 1. TPC-W workloads

2.2. Dynamic cache

Dynamic web pages are widely used in E-commerce systems. Generating these pages often requires executing queries at the database server. Thus, they cannot be cached by static caches such as web proxies. Dynamic caches, which reside on separate machines other than the database server, can store this content. They can be roughly classified as *query result cache* and *table cache*.

Query result cache In a query result cache, the query results (or the web page segments generated from these results) are saved such that subsequent identical queries can be served directly from the cache. Since the cached results may become out-of-date as the database changes, they must be invalidated. A query result cache can be easily implemented since it does not involve processing database queries. Many web server products, such as WebLogic Server JSP Cache [2], Oracle Web Cache [1], and CacheP-ortal [4], have employed query result cache. One drawback of query result cache is that it may contain duplicate data (resulted from slightly different queries).

Table cache A table cache server is a partial replication of the database server. It processes queries that access the cached tables. If a query involves uncached tables, the table cache server partitions the query into sub-queries such that some of them can still be processed locally. The database server periodically propagates relevant updates to the table cache server. Infrequently changed tables are suitable candidates for caching, so that the update propagation cost does not dwarf the benefit of caching. Since a table cache server needs to process queries, it is more complicated to implement than a query result cache. The table cache server often has much less computational power than the database server. When the database server is lightly loaded, using table cache may result in worse performance than performing all queries on the database server [10]. Currently, only a few systems use table cache, such as Oracle Internet Application Server [15] and DBCache [10].

Form-based cache Form-based cache [11] has a simple search engine that is optimized for top-n conjunctive keyword queries. This approach avoids the complexity of a full-blown query processing engine required by table cache and has better efficiency than the query result cache.

3. Experimental methodology

The system used in this study (Figure 1) is composed of *emulated browsers* (EBs), *application logic*, *dynamic cache emulator*, and *database server*, based on the TPC-W implementation at the University of Wisconsin-Madison [3].

We made extensive performance optimizations on the original implementation. We removed the web/appserver layer to speed up the emulation process, but none of these optimizations affect the databse workloads, only the time required to generate a databse request. In all experiments performed in this study, the EBs and the database server runs on one machine. This setup removes the network latency between EBs and the database server. It is also easier to automate the experiments since all components are on one machine. Preliminary studies found that the CPU, memory, and disks are not fully utilized in the experiments conducted. Therefore, this experimental setup should not change the performance of the database server.

3.1. Dynamic cache emulator

We implemented three kinds of dynamic caches: *query-result-cache*, *table-cache*, and *hybrid-cache*. We *emulated* caching functionalities rather than implementing them. We carefully designed the emulators to make sure that the workloads to the database server using these emulators are the same as those using the corresponding real caches. The dynamic caches are emulated running on separate machines.

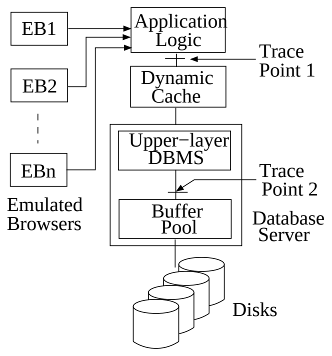


Figure 1. E-commerce system structure

Query-result-cache In the *query result cache*, cacheable queries are queries that search stable data, such as author and title search, and queries that are allowed by the TPC-W benchmark to be cached for a certain period of time, such as best sellers search. LRU is used to manage the cache. The cached results (except the author and title search results) are discarded after they stay in the cache for 30 seconds, as specified in the TPC-W benchmark. In this study, the size of the query cache is set to 20MB. Preliminary experiments found that this size is sufficiently large; cached data always becomes stale before the cache is filled.

Table-cache The *table-cache* maintains a copy for each of the infrequently-updated tables, including *country*, *item*, *author*, *address*, and *customer*. For a query that involves data only from these tables, the cache returns a faked result rather than actually processing the query. According to TPC-W benchmark, these returned results will be used only by the *product detail* web interaction to generate subsequent queries. Since all those queries involve only the cached tables, faking query results will not affect the workload to the database server. If a query involves data from tables stored only in the database server, the query is directed to database server. If a query involves data from tables stored both in cache and in the database server, it is partitioned into sub-queries, such that the sub-queries that involve only the cached tables can be processed locally by the cache. The propagation of relevant changes from the database server to the table-cache is not implemented. Since the cached tables are updated infrequently, this simplification will not noticeably impact the performance of the database server.

Hybrid-cache The *hybrid-cache* works like a table-cache, except that it also caches the query results of the *best sellers* query. This query is cacheable as specified in the TPC-W benchmark, but is not suitable to be cached by the table-cache, since it involves searching frequently-updated tables.

The cached query results expire after 30 seconds, as required by the TPC-W benchmark.

3.2. Configurations and parameters

The machine we used for the experiments is an IBM eServer xSeries 255. It has four Intel Xeon MP 1.5GHz processors with hyper-threading, 8GB RAM, and 12 34.7GB IBM U320 disks attached to two IBM ServeRAID-4Lx Ultra160 SCSI controllers. All disks are 15,000RPM with an average seek time of 3.6ms. The operating system is Windows 2000 Advanced Server, and the DBMS is IBM DB2 8.1 for Windows. The TPC-W database is built with 10,000 items and 3,000 EBs. The size of the database is 17.6GB. The page size used in the database is 4KB. The database is placed on a 5-disk hardware RAID-0.

System load is controlled by the number of EBs, which each have an exponentially distributed think time (7 seconds) between consecutive requests, as specified by the TPC-W benchmark.

3.3. Trace collection

We placed two trace points in the system, as shown in Figure 1. At trace point 1, the response time of each database query is recorded. These response times are then used to compute the database server response time for each web interaction. At trace point 2, a light-weight trace package [7] records database page references to the buffer pool.

4. Database server workload characteristics

4.1. General characteristics

Figure 2 presents the average database server response time as a function of the number of EBs. In the figure, not using any dynamic cache is labeled as *no-cache*. The response time is computed by averaging the database server response time for each web interaction. For the table-cache and hybrid-cache, this response time plus the average time required by the dynamic cache to process cached queries is the response time experienced by users. Since table cache is scalable, compared with the query processing time cost at the database server, especially heavily loaded, we suppose the query processing time cost at table cache server(s) can be negligible. So, our emulation of returning faked query results for table-cache and hybrid-cache in the experiments does not jeopardize our conclusions.

For the *browsing* and the *shopping* workloads, the no-cache system is under heavy load when the number of EBs is greater than 1600, as indicated by the rapid increase in response time. The corresponding number for the *ordering* workload is 400.

Figure 2 shows that, in general, when the system is heavily loaded, dynamic cache substantially reduces the database server response time. Under light load, query-cache has little benefit on reducing the database server response time, because a cached query result often expires before the identical subsequent query is received by the cache. The figure also shows that using dynamic cache has the largest benefit in the *browsing* workload and the smallest benefit in the *ordering* workload. This is because most queries in the *browsing* workload are cacheable, while the opposite is true for the *ordering* workload.

Since we are interested in the performance of the system when it is heavily loaded, we use workloads with 1,800 EBs in the following experiments. Figure 2 and other preliminary experiments suggest that the characteristics of the table-cache is very similar to that of the hybrid-cache. For clarity, the results of the table-cache are not shown in the remainder of this section.

Table 2 presents hit ratios of the query-result-cache and database server response time under various time-out threshold values. Using a longer time-out threshold allows the query-result-cache to absorb more queries, thus increasing hit ratios and reducing response times. These results suggest that when the database server is heavily loaded, it may be beneficial for the system to temporarily use a larger time-out threshold at the cost of more obsolete results.

Timeout (sec.)	Response Time (sec.)	Cache Hit Ratio
5	6.72	15.2%
30	4.64	40.7%
60	2.74	49.1%

Table 2. Time-out Threshold (1,800 EBs)

Table 3 shows the reductions in CPU utilization of DBMS when various dynamic caches are employed. The use of dynamic cache considerably reduces the CPU utilization, especially for the *browsing* workload, since it generates many cacheable queries. The hybrid-cache is more effective than the query-result-cache in reducing the CPU utilization.

Dynamic Cache	Browsing	Shopping	Ordering
Query-result	76.8%	20.1%	18.7%
Hybrid	89.6%	55.1%	33.9%

Table 3. DBMS CPU utilization reduction

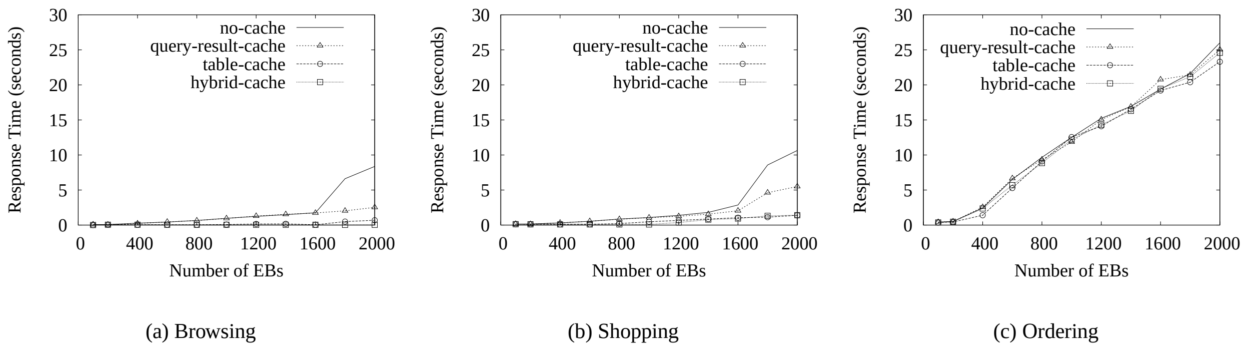


Figure 2. DB server response time (30 sec. time-out threshold)

4.2. Database page reference behaviours

In this section, we investigate database page references. Table 4 summarizes the main characteristics of the traces when using 1,800 EBs (the "Duration" is measured by minutes). Since page writes occupy only a small proportion of the total references, we consider only page reads in the following analysis. Preliminary experiments found that this omission does not noticeably influence the results.

Workload	Param.	No-cache	Query-result-cache	Hybrid-cache
Browsing	Duration	23.2	20.3	16.1
	# Requests	12.4M	1.6M	0.76M
	Read Prop.	96.6%	96.5%	92.4%
Shopping	Duration	24.3	22.0	20.3
	# Requests	10.7M	3.7M	2.6M
	Read Prop.	96.6%	96.3%	95.4%
Ordering	Duration	50.0	51.3	50.0
	# Requests	9.7M	7.8M	6.1M
	Read Prop.	95.4%	93.7%	91.9%

Table 4. Buffer pool trace characteristics

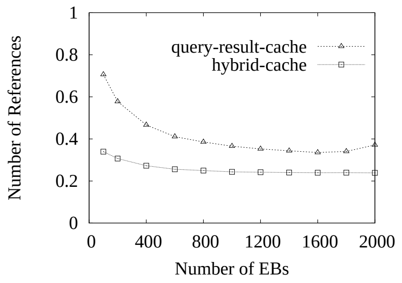


Figure 3. DB page references: *shopping*

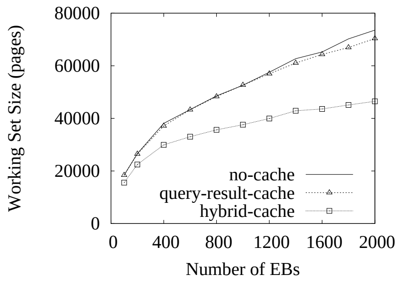
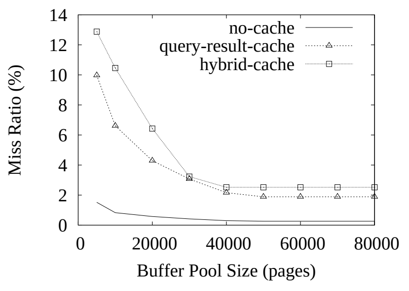


Figure 4. Shopping working set size

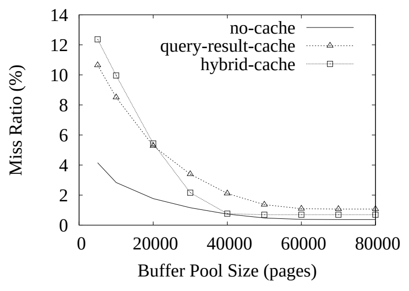
Figure 3 shows the number of database page references under the *shopping* workload when various dynamic caches are used, normalized to that without using dynamic cache. Using dynamic cache can considerably reduce the number of database page references. The *browsing* and the *ordering* workloads have similar trends and are not shown.

Figure 4 presents the number of distinct pages (i.e., the *working set* size) referenced for the *shopping* workload. The results for the *browsing* and *ordering* workloads exhibit similar trends and are not shown. The figure shows that using query-result-cache does not reduce the working set size of database page references, while using hybrid-cache can substantially reduce the working set size. This is because query-result-cache only reduces the frequency at which the cacheable queries are sent to the DBMS, while hybrid-cache reduces the number of distinct queries reaching the DBMS.

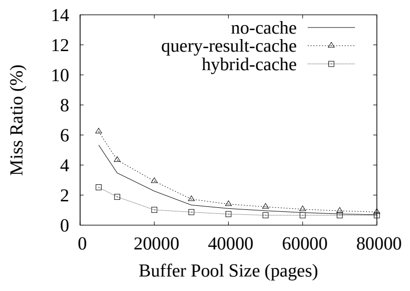
Temporal locality describes the likelihood of a page reference, given that it has been referenced recently. The buffer pool miss ratio of LRU under different buffer pool sizes (cumulative LRU stack depth [12]), can show the temporal locality. A lower miss ratio means better temporal locality in the database page references.



(a) Browsing



(b) Shopping



(c) Ordering

Figure 5. Buffer pool miss ratio

Figure 5 presents the buffer pool miss ratio when using various dynamic caches, as a function of the buffer pool size. The first half of each trace is used as the warm-up period. In Figure 5(a) and 5(b), the buffer pool has the lowest miss ratio when dynamic cache is not used. One may be tempted to conclude that the database server has the best performance when not using dynamic cache. However, as shown in Figure 3, the database server without using dynamic cache generates 2-3 times more database page references than that using dynamic cache. This larger number of references offsets all the benefits of the lower miss ratio in the buffer pool.

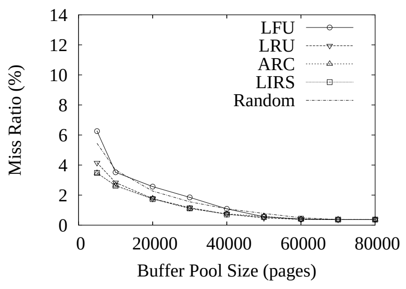
The *browsing* workload has the largest proportion of cacheable queries, while the *ordering* workload has the fewest proportion of cacheable queries. As shown in Figure 5 from left to right, in no-cache systems, the miss ratio increases as the proportion of cacheable queries in the workload decreases. This suggests that the buffer pool references generated by the cacheable queries have very good temporal locality. In other words, these queries mainly consume CPU resources, but cause few disk I/Os.

Since the query-result-cache filters out some of the cacheable queries, it always causes higher miss ratios than not using dynamic cache. Increased miss ratios are consistent with previous findings of the cache filtering effect in multi-layer file servers [17] and web proxies [6]. In Figure 5, the miss ratio gap between query-result-cache and no-cache decreases from left to right, since fewer cacheable queries are present in the workload.

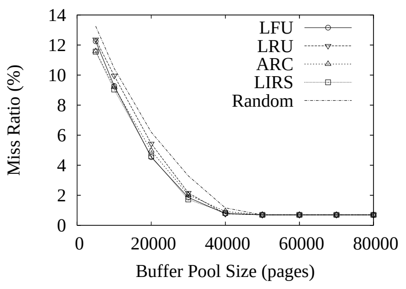
In Figure 5, the miss ratio of database page references with hybrid-cache decreases (i.e., the temporal locality increases) from left to right, as the proportion of cacheable queries in the workload decreases. In Figure 5(c), the temporal locality with hybrid-cache is better than that with no-cache. This is because the miss ratio is affected by two factors: the cache filtering effect and the working set size. The hybrid-cache filters out cacheable queries, which decreases the temporal locality of references (i.e., increases the miss ratio). On the other hand, as shown in Figure 4, the hybrid-cache has a smaller working set size than that of no-cache, providing a lower miss ratio given the same buffer pool size. In the *browsing* workload, since many queries are filtered by the hybrid-cache, the cache filtering effect dominates, giving a higher miss ratio than that of no-cache. In the *ordering* workload, since there are very few cacheable queries, the cache filtering effect almost vanishes. The smaller working set size of the hybrid-cache results in a lower miss ratio than that of no cache. The miss ratio of the hybrid-cache flattens out when the buffer pool size is larger than the working set size (about 40,000 pages as shown in Figure 4).

Two basic buffer page replacement algorithms are LFU and LRU. LFU replaces the Least Frequently Used page; LRU replaces the Least Recently Used page. ARC (Adaptive Replacement Cache)[13] automatically balances the recency and frequency of the page references. CLOCK [5] is an approximation to LRU, with the advantages of low overhead and low lock contention. LRU-2 [14] and 2Q (Two Queue) [9] both base buffer priority on sustained popularity rather than on a single access, as LRU does, while 2Q has lower overhead. LIRS (Low Inter-reference Recency Set) [8] uses recency to evaluate inter-reference recency.

Figure 6 shows the buffer pool miss ratio with various replacement algorithms. Note that ARC performs similarly to LIRS with small buffer pools, and to LRU with large buffer pools. The results for CLOCK, 2Q, and LRU-2 are not presented, since CLOCK always performs slightly worse than LRU, 2Q performs similarly to ARC, and LRU-2 performs slightly better than LIRS. Using dynamic cache causes higher miss ratios (i.e., worse temporal locality) in database page references, but to a smaller degree than that reported in file servers [17] and web proxies [6]. Although the frequency-based algorithm LFU has advantages over LRU in systems using dynamic cache, LFU performs worse than advanced recency-based algorithms, such as LIRS.



(a) No-Cache



(b) Hybrid-Cache

Figure 6. Buffer pool miss ratio (with different replacement algorithms)

Spatial locality describes how likely a page will be referenced if a page with a close-by page number is referenced. Good spatial locality in database page references implies that the buffer pool can benefit from use of a large page size and/or a prefetch policy. Spatial locality can be measured using *run length* [7]. A run length of n pages means that these n pages are accessed sequentially or almost sequentially.

Figure 7 presents the cumulative distribution of database page run lengths. For example, the point A in Figure 7(a) indicates that 40% of the references occur in sequential runs with fewer than 35 references.

Figure 7(a) shows that for no-cache running the *browsing* workload, about 50% of the references belong to sequential runs with fewer than 70 references, implying weak spatial locality. Figure 7(a) also shows that another 50% of the references belong to a sequential run of 244 references. A closer look at the trace found that this run is a sequential scan of the *author* table. Since the queries that invoke this sequential scan can be cached by the query-result-cache, a smaller proportion (20%) of references are involved in this run in the query-result-cache. This sequential run disappears in the run length distribution when using the hybrid-cache, since the *author* table is cached by the table cache server. From the left to the right of Figure 7, the proportion of references belonging to this sequential run decreases as the proportion of cacheable queries in the workload decreases.

In Figure 7, most references occur in sequential runs of small lengths, implying that prefetch may only have moderate impact on the system performance.

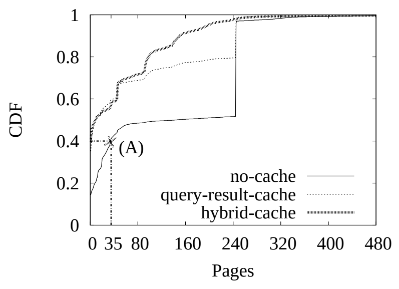
4.3. Disk I/Os

Disk I/Os are database page references that are not in the buffer pool. The number of disk I/Os is a crucial factor in performance of the database server, which is determined by many factors, including the number of database page references, the working set size of the references, the locality of the references, the replacement algorithm, and the size of the buffer pool. These factors are studied using the traces of database page references to the buffer pool. Figure 8 presents the number of disk I/Os with and without using dynamic cache, for the *shopping* workload. The results for the *browsing* and *ordering* workloads are similar and not shown. The query-result-cache generates similar number of disk I/Os to that in no-cache, unless the buffer pool is too small ($< 10,000$ pages). This implies that the database page references reduced by use of the query-result-cache are buffer pool hits in no-cache, and that the cacheable queries have a relatively small working set compared to the buffer pool size. If the working set of cacheable queries cannot fit in the buffer pool, these queries will generate disk I/Os in no-cache. In that case, using the query-result-cache will reduce the number of disk I/Os (as illustrated by the left-most point of the query-result-cache line in the figure).

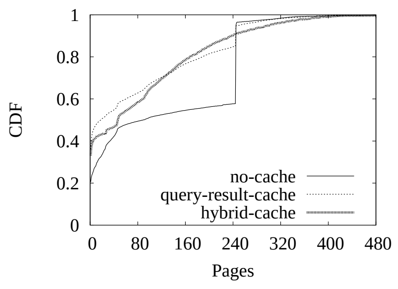
5. Discussions

Generally speaking, table cache does a better job in reducing the load on the database server than query result cache. However, a table cache server must have query processing ability, which requires higher hardware and software cost than a query result cache server. Moreover, using table cache results in longer response times than using query result cache, since the table cache needs to process each query it receives. Employing query result cache in front of table cache can accelerate query processing, but will not change the load on the database server.

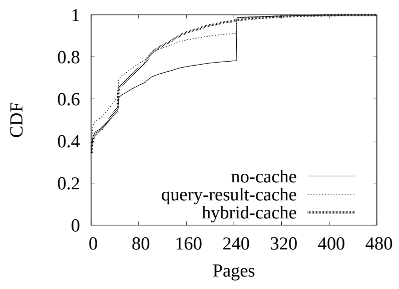
SMP (Symmetric Multi-Processor) is a commonly used technique to scale a database server. All commercial DBMSs run on large SMP servers. In a single SMP server, the memory and disks are relatively easy to expand. Several hundreds of gigabytes of memory and sev-



(a) Browsing



(b) Shopping



(c) Ordering

Figure 7. Run length of database page references

eral thousands of disks (through a dedicated storage server) can be configured. On the other hand, the scaling of processors is very difficult and costly. It is rare to see a SMP having more than 64 processors. Since dynamic cache can effectively reduce the CPU load of the database server, it can help a SMP database server support much higher load.

As E-commerce web sites provide better searching and data mining services, the proportion of cacheable queries may increase in future E-commerce workloads. Using dynamic caches may bring more benefits to such systems.

6. Conclusions and future work

The performance of the database server is crucial to an E-commerce system. The workload seen by the database server is dramatically changed by use of dynamic cache. Understanding these changes is important to the design, tuning and capacity planning of the database server.

We studied the workload characteristics of the database server in a benchmark E-commerce system. Different kinds of dynamic caches were used, including query result cache, table cache, and hybrid-cache. We found that using dynamic cache can considerably reduce the response time of the database server when it is heavily loaded. Table cache does a better job than query result cache in reducing the database server load. Using hybrid-cache only marginally further reduces the load than using table cache.

By analyzing the traces of database page references, we found that using dynamic cache can substantially reduce the number of references. The references exhibit moderate spatial locality, which can be further reduced by use of dynamic cache. In most cases, the temporal locality exhibited in these references becomes worse after using dynamic cache, but to a smaller degree than that reported in file servers and web proxies. Simple frequency-based algorithms have advantages over LRU in systems using dynamic cache. Advanced algorithms based on temporal locality, such as LIRS and LRU-2, perform the best in both systems with and without dynamic cache.

Interestingly, for workloads with few cacheable queries (e.g., the TPC-W *ordering* workload), using table cache increases temporal locality of database page references. This result is contrary to that in file servers and web proxies.

Future possible research directions include collecting and studying traces from real E-commerce systems and comparing them with TPC-W. We would also like to compare the database server workloads in E-commerce systems with traditional database workloads, i.e., OLAP and OLTP. Implementing a more realistic table cache model, which involves query processing and update propagation, and investigating its performance impact is also part of future work.

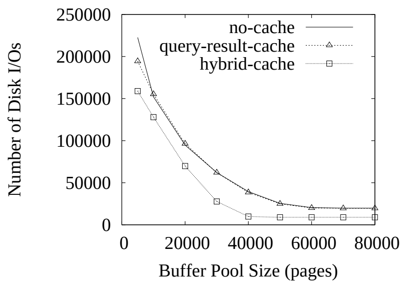


Figure 8. Number of disk I/Os

Acknowledgments

We want to thank Miso Cilimdzic for porting the trace tool to DB2 8.1, Said Elnaffar for providing the TPC-W implementation and helping the configuration of the system. We also wish to thank our system administrator Brian Gallaway for his timely help, and Rick Bunt, Derek Eager, Wei-

dong Han, Yongli An, Jianwei Song, Greg Oster, and Qing Wang for their helpful discussions and valuable comments.

References

[1] J. Anton, L. Jacobs, X. Liu, J. Parker, Z. Zeng, and T. Zhong. Web caching for database applications with Oracle web cache. In *Proc. of ACM SIGMOD'02*, pages 594–599, Madison, WI, 2002.

[2] BEA Systems, Inc. WebLogic server JSP cache. `http://e-docs.bea.com/wls/docs60/jsp/index.html`, 2004.

[3] T. Bezenek, T. Cain, R. Dickson, T. Heil, M. Martin, C. McCurdy, R. Rajwar, E. Weglarz, C. Zilles, and M. Lipasti. Characterizing a Java implementation of TPC-W. In 3^{rd} *Workshop On Computer Architecture Evaluation Using Commercial Workloads*, Toulouse, France, 2000.

[4] K. Candan, W. Li, Q. Luo, W. Hsiung, and D. Agrawal. Enabling dynamic content caching for database-driven web sites. In *Proc. of ACM SIGMOD'01*, pages 532–543, Santa Barbara, CA, 2001.

[5] F. J. Corbató. A paging experiment with the multics sytem. In *In Honor of P. M. Morse*, pages 217–228. MIT Press, Cambridge, Mass, 1969.

[6] R. Fonseca, V. Almeida, M. Crovella, and B. Abrahao. On the intrinsic locality properties of web reference streams. In *Proc. of IEEE INFOCOM'03*, San Francisco, CA, 2003.

[7] W. Hsu, A. Smith, and H. Young. I/O reference behavior of production database workloads and the TPC benchmarks – an analysis at the logical level. *ACM Trans. Database Syst.*, 26(1):96–143, 2001.

[8] S. Jiang and X. Zhang. LIRS: an effi cient low inter-reference recency set replacement policy to improve buffer cache performance. In *Proc. of ACM SIGMETRICS'02*, pages 31–42, Marina Del Rey, CA, 2002.

[9] T. Johnson and D. Shasha. 2Q: A low overhead high performance buffer management replacement algorithm. In *Proc. of VLDB'94*, pages 439–450, Santiago, Chile, 1994.

[10] Q. Luo, S. Krishnamurthy, C. M. H. Pirahesh, H. Woo, B. G. Lindsay, and J. F. Naughton. Middle-tier database caching for e-business. In *Proc. of ACM SIGMOD'02*, pages 600–611, Madison, WI, 2002.

[11] Q. Luo and J. F. Naughton. Form-based proxy caching for database-backed web sites. In *Proc. of VLDB'01*, pages 191–200, Roma, Italy, 2001.

[12] D. J. Makaroff and D. L. Eager. Disk cache performance for distributed systems. In *Proc. of ICDCS'90*, pages 212 – 219, Paris, France, 1990.

[13] N. Megiddo and D. S. Modha. ARC: A self-tuning, low overhead replacement cache. In *Proc. of the* 2^{nd} *USENIX Conference on File and Storage Technologies FAST'03*, pages 115–130, San Francisco, CA, 2003.

[14] E. O'Neil, P. O'Neil, and G. Weikum. The LRU-K page replacement algorithm for database disk buffering. In *Proc. of ACM SIGMOD'93*, pages 297–306, Washington, DC, 1993.

[15] Oracle Corporation. Oracle internet application server documentation library, 2004.

[16] TPC Benchmark^{TM} W. `http://www.tpc.org/tpcw/`, 2004.

[17] D. Willick, D. Eager, and R. Bunt. Disk cache replacement policies for network fi leservers. In *Proc. of ICDCS'93*, pages 2–11, Pittsburgh, PA, 1993.