論文 / 著書情報
Article / Book Information

Title	Cache-Conscious Data Access for DBMS in Multicore Environments
Authors	Fang Xi, Takeshi Mishima, Haruo Yokota
出典 / Citation	IEICE Transactions on Information and Systems, Vol. E98-D, No. 5, pp. 1001-1012
発行日 / Pub. date	2015, 5
URL	http://search.ieice.org/
権利情報 / Copyright	本著作物の著作権は電子情報通信学会に帰属します。 Copyright (c) 2015 Institute of Electronics, Information and Communication Engineers.

PAPER Special Section on Data Engineering and Information Management

Cache-Conscious Data Access for DBMS in Multicore Environments*

Fang XI[†a)], **Takeshi MISHIMA**[††], ***Nonmembers***, ***and*** **Haruo YOKOTA**[†], ***Fellow***

SUMMARY In recent years, dramatic improvements have been made to computer hardware. In particular, the number of cores on a chip has been growing exponentially, enabling an ever-increasing number of processes to be executed in parallel. Having been originally developed for single-core processors, database (DB) management systems (DBMSs) running on multicore processors suffer from cache conflicts as the number of concurrently executing DB processes (DBPs) increases. Therefore, a cache-efficient solution for arranging the execution of concurrent DBPs on multicore platforms would be highly attractive for DBMSs. In this paper, we propose CARIC-DA, middleware for achieving higher performance in DBMSs on multicore processors, by reducing cache misses with a new cache-conscious dispatcher for concurrent queries. CARIC-DA logically range-partitions the dataset into multiple subsets. This enables different processor cores to access different subsets by ensuring that different DBPs are pinned to different cores and by dispatching queries to DBPs according to the data-partitioning information. In this way, CARIC-DA is expected to achieve better performance via a higher cache hit rate for the private cache of each core. It can also balance the loads between cores by changing the range of each subset. Note that CARIC-DA is *pure* middleware, meaning that it avoids any modification to existing operating systems (OSs) and DBMSs, thereby making it more practical. This is important because the source code for existing DBMSs is large and complex, making it very expensive to modify. We implemented a prototype that uses unmodified existing Linux and PostgreSQL environments, and evaluated the effectiveness of our proposal on three different multicore platforms. The performance evaluation against benchmarks revealed that CARIC-DA achieved improved cache hit rates and higher performance.

key words: *multicore, OLTP, middleware, cache*

1. Introduction

The continued evolution of modern hardware has brought several new challenges to database (DB) management systems (DBMSs). Recently, microprocessor manufacturers have found it increasingly difficult to make CPUs operate faster owing to size, complexity, clock skew and heat issues. Thus, they are continuing the performance curve by placing multiple CPUs on a single chip and relying on parallelism to obtain higher performance gain, which has brought the computing world into the so-called multicore era. Meanwhile, main memory sizes have rapidly increased along with microprocessor performance, and the memory plays the role of a disk in many applications whose working sets fit in the memory. However, the data-intensive application of DBMSs cannot fully take advantage of the availability of the numerous cores. This is largely due to the fact that processor clock rates have improved much faster than memory latency, leading to the situation that the program data cannot be delivered fast enough to be consumed by the numerous processor cores. Traditional DBMSs, which are dedicated to improving database performance through I/O optimization, fail to utilize processor resources efficiently. Therefore, exploiting the characteristics of modern multicore processors has become an important topic of database system research [1], [2].

Cache memories are intended to contain copies of main memory blocks to speed up access to frequently needed data. The cache levels are critical for overcoming the "memory wall" in DBMS applications [3]. Ailamaki *et al.* [4] analyzed the memory hierarchy performance of commercial DBMSs and pointed out the importance of the last-level cache (LLC). Follow-up studies on PAX [5] and CSB-Tree [6] addressed the problem by providing cache-efficient data structures for DBMSs. However, these studies only addressed the problem for single-process execution models and did not solve the orthogonal problem of how to improve cache performance for concurrent queries. A study on MCC-DB [7] pointed out the conflicts in the shared cache for concurrent queries on the multicore platform and solved the problem by integrating the OS facility of cache partitioning into DBMS systems.

As integrated circuits become denser, designers have larger chip areas that can be devoted to on-chip caches. For modern multicore processors, it is usual to provide two levels of cache for the private use of each processor core (private-cache levels) in addition to a shared LLC. For example, the AMD Opteron 6174 processor [8] provides two levels of private cache for each core, namely a level-one (L1) instruction cache, an L1 data cache, and a level-two (L2) cache. As the cache levels become more complex and the last-level cache size is scaled, the access to the LLC involves an increasing number of clock cycles. These changes in cache levels indicate that it is increasingly important to bring data beyond the LLC and closer to L1. Hardavellas *et al.* [9] proposed STEPS [10], which is a transaction-coordinating mechanism that minimizes instruction misses in the L1 cache based on the StagedDB design. However, reducing the data misses in higher cache levels is still a major challenge. Therefore, in this paper, we first analyze how var-

Manuscript received July 25, 2014.
Manuscript revised November 28, 2014.
Manuscript publicized January 21, 2015.

[†]The authors are with the Department of Computer Science, Tokyo Institute of Technology, Tokyo, 152–8552 Japan.
[††]The author is with Software Innovation Center, NTT Japan, Musashino-shi, 180–8585 Japan.

*This is a continuation and extension of our work proposed in DASFAA 2014 [1] and FTSIS 2014 [2].

a) E-mail: xifang@de.cs.titech.ac.jp

DOI: 10.1587/transinf.2014DAP0004

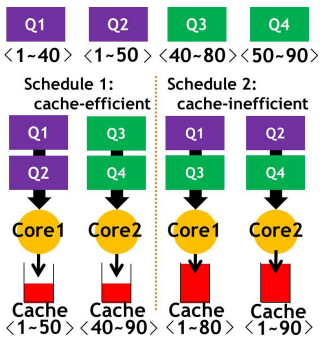

Fig. 1 Different query-dispatching strategies and related private-cache performance.

ious scheduling strategies for concurrent DB processes on different processor cores affect the performance of private-cache levels, which are closer to the execution unit than the LLC. We then propose a middleware-based system to provide efficient data access to the private-cache levels for concurrent OLTP-style transactions on multicore platforms.

A typical OLTP workload consists of a large number of concurrent short-lived transactions, each accessing a small fraction of a large dataset. The most straightforward and simplest way for DBMSs to utilize the available parallelism provided by a multicore system is to deal with many concurrent OLTP requests simultaneously [11]. Furthermore, all concurrent DB processes (DBPs) dealing with various queries should be dispatched to run concurrently on different processor cores. The several DBPs dispatched to the same processor core will then be executed alternately, with each DBP reusing the cache lines loaded by its forerunner. However, a different DBP-dispatch decision will lead to different cache performance. If the queries that access data in the same subset co-run on one core, the cache hit rate increases. For example, Fig. 1 shows two query-dispatching strategies for four concurrent queries on two processor cores. In the cache-efficient solution (Schedule 1 in Fig. 1), the two queries that access the same data are dispatched to run on one processor core. Q1 can reuse the cache data $\langle 1-50 \rangle$ which are already loaded into the cache after the execution of Q2. However, in the cache-inefficient solution (Schedule 2 in Fig. 1), Q1 runs after Q3, and it cannot reuse the cache data loaded by Q3. There will be more cache misses in the cache-inefficient solution than in the cache-efficient solution. Furthermore, the cache-efficient solution can restrict the data access for the private cache of each core to within a smaller subset, with the probability of cache hits thereby being increased. In contrast, the cache-inefficient solution, where the cache, accesses a much larger dataset, can easily cause cache misses in the private cache, which are usually small (several hundred KB).

Motivated by this example, we analyzed the possibility of achieving higher private-cache utilization through optimized dispatching for concurrent queries. We proposed a framework for cache-conscious data access for OLTP applications on multicore platforms by combining some functions of the operating system (OS) and the DBMS. As cache optimization can be both daunting and complex, researchers usually have to redesign the algorithms and data structures carefully in accordance with the underlying hardware. In contrast, our proposed system can be implemented as middleware and can provide significant performance improvement on different modern multicore platforms. Our work makes the following contributions:

- We propose a framework—Core Affinity with Range Index for Cache-conscious Data Access (CARIC-DA) to improve the private-cache performance for DBMSs on multicore platforms. CARIC-DA dispatches all queries accessing data in the same value range for execution by a specific DBP. Meanwhile, the core-affinity function provided by the OS is used to ensure that each DBP always runs on a specific processor core. To our knowledge, CARIC-DA is the first middleware addressing the data miss problem in private-cache levels for concurrent queries.
- A major feature of CARIC-DA is that it is *pure* middleware, and all the functions provided by CARIC-DA can be implemented as middleware over existing OSs and DBMSs. Our system can be easily applied to existing DBMS systems as it does not require any modification to existing OSs and DBMSs. This is important because software for existing DBMSs is usually very large and complex, and any modification would be time-consuming and difficult.
- As multicore platforms have become increasingly diverse and complex in recent years, it is becoming difficult for software to benefit from different multicore platforms. We compared the performance of our proposed system with a pure PostgreSQL system across three different multicore platforms (an AMD platform and two Intel platforms) and obtained several insights into how individual cache levels contribute towards improving the performance of modern multicore platforms.
- Our experiment results verified the efficiency of CARIC-DA in improving the performance of DBMS applications on different multicore platforms. Our system can improve the throughput by 21% and 28% on the two Intel platforms. On the AMD platform, CARIC-DA can reduce the L2 cache miss rate by 56% and the L1 cache miss rate by 6–10%. In addition, it can increase the throughput by up to 25% for a TPC-C workload [12].

2. CARIC-DA Framework

When a database system is presented with multiple queries, opportunities arise for optimizing the group of queries as a whole. As different queries have different data requirements, which determine how much a query can benefit from accessing accumulated private-cache data, we consider dispatching the queries to run on different processor cores ac-

cording to the data requirements of different queries. To provide efficient private-cache utilization for each processor core, the queries that access the same dataset should be dispatched to run on the same core. Meanwhile, we should dispatch the queries that access data in different value ranges to run on different processor cores. On the basis of these principles, CARIC-DA dispatches the queries to co-run on different processor cores based on a logical partition of the whole dataset in the database. Firstly, CARIC-DA logically partitions the whole dataset in the database into several subsets, and secondly, CARIC-DA dispatches the queries accessing the data in a specific subset to run on a specified processor core.

The DBMS has the information of the whole dataset and the data-accessing information about different DBPs. However, the DBMS cannot ensure that specific DBPs run on specified processor cores, as the scheduling of DBPs on processor cores is decided by the OS. One straightforward approach would be to change the processing scheduling strategy in the existing OS. However, considering the complexity of existing OSs, this would be impractical. CARIC-DA offers a more practical approach with no modification needed in either the DBMS or the OS. It achieves its goal by a two-step strategy.

(1) Dataset and DB-process binding: The first step is for CARIC-DA to associate each DBP with a disjoint subset of the DB and to ensure that queries that access data in the same subset are executed by the same DBP. As an example, consider a table with 3k lines. Suppose that there are three DBPs, namely DBP1, DBP2, and DBP3, in our system. Each DBP can access a disjoint subset of 1k lines, for example, DBP1 accesses the lines numbered 1–1k. All queries that access lines numbered 1–1k will be dispatched for execution by DBP1, which can access the data in this subset, but queries requiring data for line number 1500 will be assigned to DBP2.

(2) DB-process and processor-core binding: Second, we aim to force each DBP to run only on a specific processor core by setting the CPU affinity for each DBP. The core affinity setting is achieved via a function provided by the Linux OS called the CPU affinity. For example, if a CPU affinity of 1 is set for DBP1, this DBP will always run on processor core 1 and will never be scheduled for another processor core by the OS.

In this way, we reach the goal, namely, binding between datasets and cores. In the example, all queries that access lines numbered 1–1k are dispatched to run on processor core 1. In addition, all queries accessing data in different subsets are dispatched to run on other processor cores.

2.1 Logical Partitioning of Database

The main components of CARIC-DA are shown in Fig. 2. CARIC-DA extends an existing DBMS by introducing a middleware subsystem. The binding between the dataset and the DBP is achieved by logical partitioning of the whole database. The mapping information for coupling different partition key ranges with different DBPs is formulated and stored in CARIC-DA as the range index (RI).

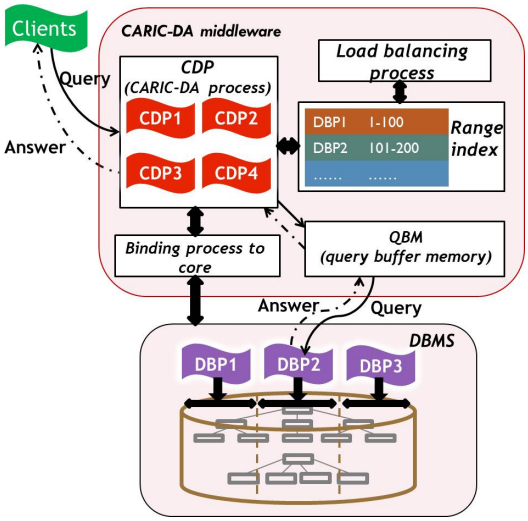

Fig. 2 Application of the CARIC-DA framework to conventional DBMSs.

A horizontal partition is adopted for each table and a suitable field for a table is chosen as its partition key. As this is a logical partition, the indexes that are originally built on large tables do not need to be changed. The table data and the related part of the B+ tree are assigned to the same logical partition. If there are multiple tables, the various tables are partitioned or overlapped, and mapped separately to all DBPs. That is, each process is mapped to several subsets for the different tables. Theoretically, any field of a table can serve as a partition key. However, in practice, the fields of the primary key of the table or only a subset of them can achieve good results, as we show in our experiments. For multiple tables, it is better to choose keys that can be commonly used to well partition most of the main tables (comparatively large and frequently accessed tables). For example, the primary key of the Customers table of the TPC-C database consists of the Warehouse id, the District id, and the Customer id. The partition key fields may be the Warehouse id and the District id. These can also be used to well partition the other main tables. It is not necessary to divide all the tables in the database as there is a trade-off between the cache hit gain and the query transfer cost.

The CARIC-DA processes (CDPs) use the mapping information provided by the RI to dispatch queries to DBPs with different data requirements into an appropriate buffer in the query buffer memory (QBM). There is no physical partitioning of the DB or changes to the original DBMS, because the mapping work is performed by the CDPs, i.e., decomposing and dispatching the various queries according to the RI while considering access skews. Instead of obtaining queries directly from clients, each DBP only deals with queries in a specific buffer where the queries access only a specified dataset.

2.2 Binding Process to Core

The ability for an OS to bind one or more processes to one or more processors is called CPU affinity [13]. One of the benefits of CPU affinity is the optimization of cache performance. As scheduling a software process to be executed on a specified processor core could result in the efficient use of a processor by reusing preloaded resources such as the data in the cache levels, many OSs, from Windows Vista to Linux, have already provided a system call to set the CPU affinity for a process. On Linux, the CPU affinity of a process can be altered by the task set program and the sched_setaffinity system call. We use sched_setaffinity to set the affinity and give a different "bitmask" to different processes. For example, if the bitmask is set to 16, the process can only run on processor core number 4. As the decimal number 16 can be converted to the binary number 10000_2, the fifth bit is set to 1, therefore the process can run on processor core number 4.

In our framework, we set the CPU affinity for both the DBPs and the CDPs. For data-intensive applications, the DBPs have large data-access requirements and mainly access the data in the DB. The CDPs frequently access the RI and QBM. We therefore bind the CDPs and DBPs that access different datasets to different processor cores to avoid cache-pollution problems.

2.3 Query Processing

In our CARIC-DA framework, CDPs communicate with the clients and dispatch the queries to different DBPs. For example, in Fig. 2, CDP1 receives a "select" query from a client. CDP1 first checks the RI to identify the appropriate mapping between the dataset related to this select query and the DBPs. Suppose DBP2 can access the dataset required by this query. The query will then be placed in Buffer2 in the QBM for checking by DBP2. Whenever DBP2 detects a query request, it picks up the query from Buffer2. The select query is executed and the answer is placed back in Buffer2 by DBP2. CDP1 then transfers the answer in Buffer2 back to the relevant client.

When dealing with a range query that accesses a large data range, the query request can be divided into several subqueries that are transferred to several different buffers. If queries require data that are mainly mapped to one DBP but have a small portion (several lines of a table) mapped to another DBP, we may prefer to dispatch this kind of query to one DBP instead of two different DBPs. This is because the query-and-answer transfer is an additional overhead for our platform compared with traditional DBMS implementations. At the system level, if the query-dispatching function provided by the CARIC-DA framework can bring about a sufficient improvement to the whole system, the query transfer will be worth implementing. We can therefore assume that a small imbalance will not affect the overall performance, because the imbalance will relate to only a few lines of the table and might not justify the extra query-transfer cost.

Dealing with cross-partition transactions is a challenge for our system as not all the datasets can be well partitioned in many applications. For transactions with lower isolation requirements, our middleware can decompose one transaction into several sub-transactions and dispatch them to different partitions. However, we avoid transactions that require a very high isolation level to cross different partitions. For the TPC-C benchmark, we assign the New-order transaction to a specified partition according to the District id and leave the Item table unpartitioned.

Algorithm 1 Calculation of new value ranges for DBPs

Create N_{sub} subsets by dividing sets with $l_i > l_{ave}$ into subsets
Let SN be the serial number of subsets, $SN \leftarrow 1$
for i in $[1, N]$ **do**
  if $l_i > l_{ave}$ **then**
    divide the dataset into μ subsets where $\mu \leftarrow \lceil \frac{l_i}{l_{ave}} \rceil$;
    calculate the load and range for each subset SN as $sl_{SN} \leftarrow \frac{l_i}{\mu}$, $sR_{SN} \leftarrow \frac{R_i}{\mu}$, $SN \leftarrow SN + 1$;
  else
    $sl_{SN} \leftarrow l_i$, $sR_{SN} \leftarrow R_i$, $SN \leftarrow SN + 1$;
Merge N_{sub} subsets into N sets with $l_i = l_{ave}$, $j \leftarrow 1$
for i in $[1, N]$ **do**
  $l_i \leftarrow l_{ave}$, $R_i \leftarrow 0$
  while $l_i > sl_j$ **do**
    merge subset j into set i ($R_i \leftarrow R_i + sR_j$), $l_i \leftarrow l_i - sl_j$, $j \leftarrow j + 1$
  if $l_i > 0$ **then**
    merge a part of subset j into set i ($R_i \leftarrow R_i + \frac{l_i}{sl_j} \times sR_j$), $sR_j \leftarrow 1 - \frac{l_i}{sl_j} \times sR_j$

2.4 Load Balancing

CARIC-DA attempts to balance the load between DBPs by modifying the ranges of values mapped to different DBPs and to synchronize the CDPs dispatching the queries on the basis of the new dataset-mapping information of the RI. The load-balancing process regularly gathers information on the number of queries processed by each of the N DBPs as a measure of the load on each DBP (l_i). This information-gathering process has negligible as it only reads several variables. Furthermore, the time interval between two consecutive gathering and load-balancing processes can be modified by the user according to their applications, and we set this variable to 60 s in our experiments. The load-balancing process calculates the sum of l_i, denoted as l_{sum}. The DBPs with the largest load and the smallest load are identified. The load difference is calculated as $l_{diff} = l_{large} - l_{small}$ and the skew is defined as $skew = \frac{l_{diff}}{l_{sum}}$. Tolerance a small load imbalance is achieved by omitting the rebalancing process whenever $skew$ is below a threshold. When the skew is larger than a threshold, Algorithm 1 is invoked to calculate new datasets (R_i is the range of the ith dataset) for the DBPs. To describe the algorithm, let l_{ave} be the ideal load for each partition. Then, the ideal load for each partition is $l_{ave} = \frac{l_{sum}}{N}$. In the first part of the algorithm, we create N_{sub} subsets by dividing

the ranges with $l_i > l_{ave}$ into subsets. We denote the load and range of each subset as sl_i and sR_i, respectively. We ensure that $sl_i < l_{ave}$ during the partitioning process and calculate the data range sR_i for the subsets. In the next part of the algorithm, we merge these subsets into N datasets that have $l_i = l_{ave}$. The new dataset information is written back to the RI, and all CDPs are synchronized to dispatch the queries according to the new RI.

2.5 Applications

CARIC-DA is designed to optimize the performance for OLTP systems. Therefore, probably not all workloads can significantly benefit from our proposal. A typical OLTP workload consists of a large number of concurrent, short-lived transactions, each accessing a small fraction (up to tens of records) of a large dataset. The smaller cache footprints of these transactions make data sharing between sequences of transactions possible in the private-cache levels, which are relatively small. In contrast, online analytical processing (OLAP) applications with queries involving aggregation and join operations on large amounts of data cannot benefit from our middleware. The private data structures during the query execution process are relatively large, such as a hash table for the hash join operator, and these data can easily occupy all of the private cache. However, these private data cannot be reused by the following queries. Even though joins and aggregations can be partitioned to smaller subqueries, considering the large data size of OLAP applications, the private data for subqueries are still too large for the private-cache levels. Although this may appear a limitation, this is not the case in the context of database applications as there is no optimal solution for all applications [14], [15].

In the following part, we explain the performance of CARIC-DA based on traditional relational database management systems (RDBMSs). However, the application of CARIC-DA is not restricted to RDBMSs. Recent key-value stores [16]–[18] provide high performance, partially by offering a simpler query and data model than RDBMSs. The key-value databases can also benefit from our proposal. However, to limit the length of this paper, we only analyze CARIC-DA with RDBMSs in this paper.

3. Performance Evaluation

The CARIC-DA-related functions were implemented in the C language as a middleware subsystem over the existing PostgreSQL [19] and Linux. In this section we compare the CARIC-DA–PostgreSQL-based system with an unmodified PostgreSQL (baseline) system on multicore platforms. We measured the response time, throughput, and cache performance of each system for different clients, platforms, and database sizes. We performed micro-benchmark evaluation to isolate the effects and to provide an in-depth analysis, and then we used the more complex and realistic workload of the TPC-C benchmark to further verify the effectiveness of our proposal.

Table 1 DB Server Parameters.

Processor	Intel(Xeon)		AMD(Opteron)
	E5-4650	E7-4860	6174
Sockets	4	4	4
Cores/Socket	8	10	12
Frequency	2.7GHZ	2.26GHZ	2.2GHZ
HW Contexts	64	80	48
L1D(per core)	32KB	32KB	64KB
L1I(per core)	32KB	32KB	64KB
L2(per core)	256KB	256KB	512KB
L3/LLC(shared)	20MB	24MB	12MB(2×6MB)
Memory	64GB	32GB	32GB

We set the value of $shared\_buffers$ to 20 GB for PostgreSQL, and this setting ensured that all DB tables in the following experiments could fit in the main memory, enabling I/O contention to be taken off the list of critical issues in our experiments. We used the profile tool Oprofile [20] to examine the cache performance [21], [22].

3.1 Hardware Platforms

We evaluated the performance of CARIC-DA on three different multicore platforms, and Table 1 describes the AMD and Intel platforms used for the DB server in detail. The two Intel platforms have private-cache levels of the same size while they are different in many ways, such as the number of cores per socket, the frequency, and the size of the LLC. In our experiments, we always enabled the function of Hyper-Threading for the Intel platforms, which enabled us to have two logical CPUs (virtual CPUs) on one physical core. The AMD platform does not have the Hyper-Threading function, but there are some node structures in each socket. Additionally, the cache levels are also different between the Intel and AMD processors. As it is well known that the cache performance is closely related to the size of the caches, it is essential to compare the performance on these two types of multicore platforms. We set up the baseline system and our CARIC-DA system on the DB servers. The clients run on four separate machines (each had an Intel Xeon E5620 CPU) and communicate with the DB servers through network connections.

3.2 Benefits of CARIC-DA

In this section, we describe the benefits of CARIC-DA compared with the baseline system on a modern AMD multicore platform. We analyzed the optimal configurations of CARIC-DA considering the node structure of the AMD platform. Then we compared the performance of CARIC-DA with the baseline system for different aspects.

3.2.1 Core Affinity in the System

On the AMD platform, there are four processors, and each processor is a package that contains two nodes. A node is an integrated circuit device that includes (1) six cores, (2) links for communication to other devices, (3) DDR DRAM

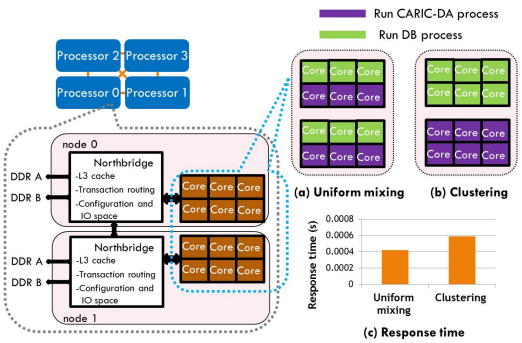

Fig. 3 Core affinity in the system.

interfaces, and (4) one northbridge (NB) including an L3 cache [23]. There are two types of processes (the CARIC-DA and DB processes) in our proposed system, and how to bind these processes to different processor cores is a problem as inappropriate binding strategies will cause competition or waste hardware resources.

We defined two core-affinity strategies, a uniform mixing strategy and a clustering strategy, and compared their performance. Under the uniform mixing strategy, a mixture of several DB processes and CARIC-DA processes is located in one node structure (Fig. 3 (a)). Under the clustering strategy, all the DB processes are bound to cores located in node0 and all the CARIC-DA processes are bound to cores located in node1 (Fig. 3 (b)).

The average response times under the two core-affinity strategies are shown in Fig. 3 (c). The uniform mixing strategy performs better than the clustering strategy. In the clustering strategy, all data-intensive DB processes are bound to the same node, leading to intensive competition for the shared node resources (L3 cache and memory interfaces). Therefore, we used the uniform mixing strategy for the CARIC-DA system in subsequent experiments.

3.2.2 Efficiency of the CARIC-DA System

We initialized the database with a 100,000-line stock table from the TPC-C benchmark and compared the performance of the CARIC-DA system with the baseline system using the micro-benchmark with each transaction randomly accessing one line of the table. We evaluated the CARIC-DA system with settings of 3, 6, 12, and 24 CDPs. For each setting we set up 3, 6, 12, and 24 clients accordingly, with different clients connected to different CDPs. In the baseline system, the clients were connected directly to PostgreSQL. The response times for different systems are shown in Fig. 4 (a). The scalability of the 24 CDP system is shown in Fig. 4 (b).

The AMD platform benefits greatly from CARIC-DA. When there are 24 concurrent clients, the select-intensive transactions are executed 33% faster under CARIC-DA (Fig. 4 (a)). Moreover, the CARIC-DA system can greatly improve the throughput by 48% (Fig. 4 (b)).

We repeated the experiment for the 24 CDP system with 48 concurrent clients and separately measured the cache miss rate for various cache levels to confirm that

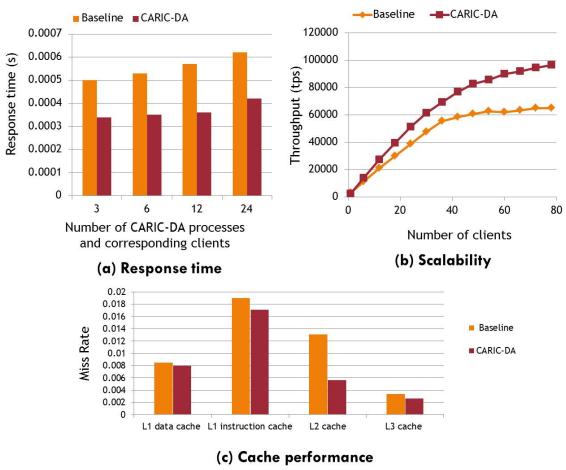

Fig. 4 Advantage of CARIC-DA system.

CARIC-DA is efficient because of its outstanding performance at various cache levels. The miss rate is the number of misses as a percentage of the total number of instructions. Figure 4 (c) shows that the CARIC-DA system can achieve almost 56% reduction in L2 cache miss rate compared with that for the baseline system. The CARIC-DA system can also improve the performance of the L1 instruction cache by up to 10%. This is because all the DBPs are restricted to running on specific processor cores in the CARIC-DA system, which can avoid frequent context switching in the cores.

We measured the time cost in transferring data between CDPs and DBPs, which is an overhead of our middleware compared with the baseline system. It took 0.06 ms for each query in the 24 CDP system. Even though the framework has to devote extra time to dispatching queries, the CARIC-DA system can achieve much better overall performance than the baseline system by greatly improving cache utilization.

3.3 Performance of CARIC-DA System on Different Multicore Platforms

Multicore platforms have become increasingly diverse and complex in recent years, and manufacturers are making various efforts to boost their performance. AMD and Intel are two major microprocessor manufacturers that use very different techniques in developing multicore processors. There are no node structures on the chip of an Intel multicore processor, and Intel processors support the Hyper-Threading function. On the other hand, the size of the cache levels is also greatly different for AMD and Intel multicore processors. In this subsection, we further analyze how the different platforms affect the database systems, and we reevaluate the efficiency of our CARIC-DA framework on two modern Intel platforms.

3.3.1 Different CARIC-DA Systems

On the Intel platform, the Hyper-Threading function enables

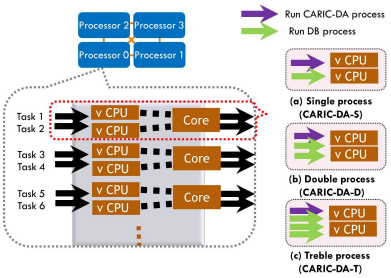

Fig. 5 Different CARIC-DA systems.

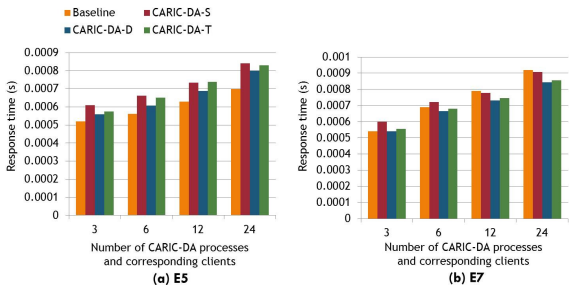

Fig. 6 Effectiveness of the CARIC-DA framework on Intel platforms.

multiple threads to run on each physical core, and as a performance feature, it also increases processor throughput. As shown in Fig. 5, two virtual CPUs correspond to one physical core. Better performance is more likely to be achieved by assigning more than one context to one processor core on the Intel platform. However, too many processes concurrently running on one core may result in low performance because these processes will compete for CPU resources. Therefore, we consider several different CARIC-DA systems and bind a different number of processes to one core. For the CARIC-DA-S system, we create one DB process for each CARIC-DA process and assign one CARIC-DA process and one DB process to two virtual CPUs corresponding to the same core (Fig. 5 (a)). For the CARIC-DA-D system, we create two DB processes for each CARIC-DA process and assign one CARIC-DA process and two related DB processes to two virtual CPUs (Fig. 5 (b)). For the CARIC-DA-T system, we create three DB processes for each CARIC-DA process and assign one CARIC-DA process and three related DB processes to two virtual CPUs (Fig. 5 (c)).

3.3.2 Response Time

We recompared the performance of the CARIC-DA systems with the baseline system on two Intel platforms using the same database and workload as in earlier experiments. The response times for different systems are shown in Fig. 6.

For the baseline system, there is a greater decrease in performance on the Intel platforms upon setting up more concurrent clients than on the AMD platform. The response time of the baseline system is approximately 0.5 ms when there are three concurrent clients on all three different platforms. However, it increases to 0.7 and 0.9 ms when there

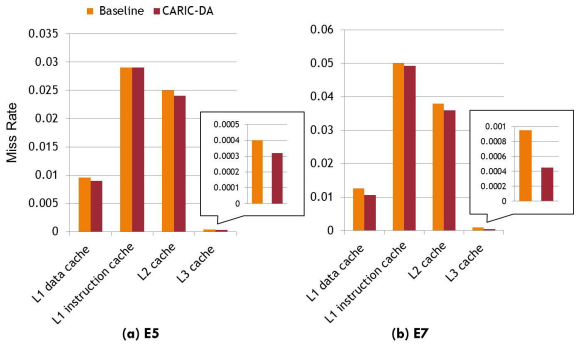

Fig. 7 Cache miss rate on different platforms.

are 24 concurrent clients on the Intel platforms, while it is 0.6 ms on the AMD platform. These results indicate there is more intensive competition for hardware resources (in the cache, interconnections, and so forth) for concurrent database processes on the Intel platforms than on the AMD platform.

For the CARIC-DA system, the advantages are less evident on the Intel platforms than on the AMD platform. The CARIC-DA-D system achieved the best performance. Two DBPs running on one physical core is efficient as this setting can take advantage of the Hyper-Threading function. However, none of the CARIC-DA systems achieved as large performance improvement on the Intel platforms as on the AMD platform. We observed that the overhead of our middleware is higher on the Intel platforms, and it is approximately 0.128 ms for the 24 CDP CARIC-DA-S system. Moreover, our proposal of reducing data cache conflict in the private-cache levels has less effect on the Intel platforms than on the AMD platform. This can be observed from the calculated cache performances for different systems on the Intel platforms analyzed in the next subsection.

3.3.3 Cache Performance

We further analyzed the cache performance on the Intel platforms. We separately measured the cache miss rate when there were 48 concurrent clients for the CARIC-DA-S system with 24 CDPs and the baseline system.

As shown in Fig. 7, the cache performance is improved slightly on the two Intel platforms, while on the AMD platform, the proposed middleware markedly reduces the cache miss rate for different cache levels. On the Intel platforms, the L2 cache performance is improved by 4% and 5.2% for the two systems (Figs. 7 (a) and (b)). The L2 cache size is 256 KB on the Intel platforms and 512 KB on the AMD platform. Also, the L1 cache of the Intel platforms is also much smaller than that at the AMD platform. Therefore, on the Intel platform, even though the private cache of each core accesses a smaller dataset in the CARIC-DA system than in the baseline system, the compared smaller dataset is still much larger than the size of the private caches. This is one reason why the CARIC-DA system performs better on the AMD platform.

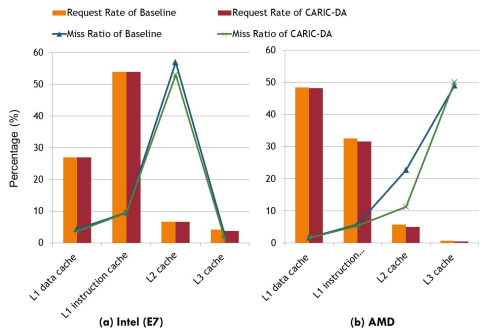

Fig. 8 Cache miss ratio and request rate of different platforms.

Comparing the two Intel platforms, the Intel E5 platform benefits less from CARIC-DA than the Intel E7 platform on the LLC, as the LLC is larger on E7 than on E5. Every time there is an LLC cache miss, the processor requires much more time to access the data in the memory space, and the memory access is at least 10 times slower than the cache access to the LLC [24]. Moreover, the overhead of the middleware cannot be compensated by the reduction in cache contentions on the E5 platform. Therefore, CARIC-DA had inferior performance to the baseline system on the E5 platform.

We also monitored the cache miss ratio (the number of misses as a percentage of the total number of cache requests) and the cache request rate (the number of requests to different caches as a percentage of the total number of instructions) for different cache levels.

As the cache performance is similar for the two Intel platforms, we use the performance of the Intel E7 platform to represent that of the Intel platforms. From the cache request rate of the two platforms, we found that the performance of the private-cache levels is dominated by instruction misses on the Intel platform while the AMD platform is mainly affected by data misses. On the Intel platform, there are more instruction requests to the L1 instruction cache than on the AMD platform (Fig. 8). Furthermore, we found that more than 50% of the missed L2 cache requests on the Intel platform were instruction fetch requests. On the Intel platform, the much longer instruction footprint of the DB process cannot be cached by the smaller L1 instruction cache, and L1 instruction misses dominate the performance in the cache levels. The smaller private-cache size and the higher rate of instruction cache misses mean that CARIC-DA cannot achieve a significant improvement in data access on the Intel platform. On the other hand, our middleware can reduce the L3 cache miss ratio by 50% on the Intel platform as the L3 cache of the Intel platform is two times larger than the L3 cache of the AMD platform (Fig. 8 (a)).

As the physical designs of the cache levels are different for different multicore platforms, different platforms benefit differently from our cache-conscious data access. The AMD platform has a much larger L2 cache, therefore, it can benefit significantly from our proposal. On the other hand, the Intel platform with a smaller private cache and a high rate of instruction misses cannot benefit significantly from our proposal. These results highlight the fact that as hardware becomes diverse, the optimization of software has to be carried out with close attention to features of different hardware.

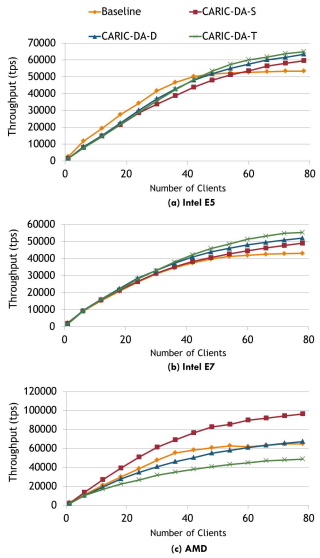

Fig. 9 Scalability of different CARIC-DA systems.

3.3.4 Throughput

In this subsection, we give a detailed analysis of the performance of the CARIC-DA systems under various concurrent query loads on the three platforms. We set up CARIC-DA systems with 24 CDPs on the DB-server platforms and repeated the micro-benchmark experiment. The changes in the throughput are shown in Fig. 9.

The default scheduling of the operating system cannot fully use the available parallelism of the numerous cores and the baseline system suffers from the non-scalability problem on all three platforms, while our CARIC-DA system achieved higher throughput on all three platforms. On the two Intel platforms, the CARIC-DA-T system achieved the highest throughput, and for 78 concurrent clients, the throughput of the CARIC-DA-T system was 21% and 28% higher than that for the baseline system. However, the CARIC-DA-S system had the best performance on the AMD platform. On the other hand, our CARIC-DA system is also nonlinear. For the CARIC-DA-S system, the concurrent queries accessing data in the same subset are dispatched to the same DB process. The DB process has to process these queries one by one, and there will be some wait time in the buffer for these queries. This is one reason for the nonlinear throughput of our CARIC-DA systems. On the other hand, the CDP generates one client thread to deal with the queries in one client connection, and we use the System-V semaphores to synchronize the accesses to the QBM by these threads. With an increasing number of concurrent clients, any untimely scheduling of these semaphores will

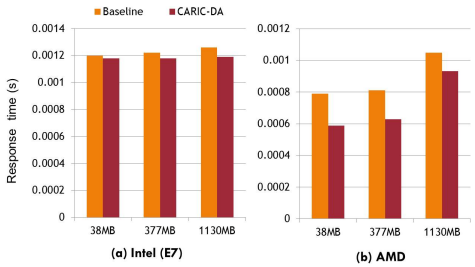

Fig. 10 Response time for datasets of different sizes.

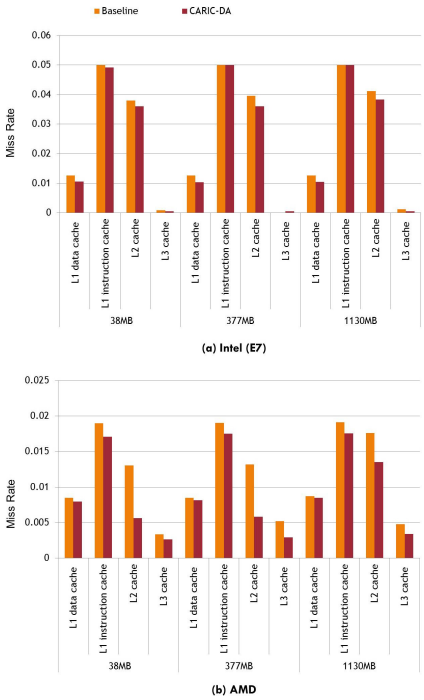

Fig. 11 Cache performance for datasets of different sizes.

lead to diminished performance.

On the Intel platform, increasing the number of DB processes on each processor core does not introduce resource contention, and the difference in performance between different CARIC-DA systems is not significant. However, we proposed the binding of one DB process to one core on the AMD platform as we observed a significant decrease in performance for the CARIC-DA-D and CARIC-DA-T systems. Therefore, in the following subsections, we use the CARIC-DA-S system as the default setting for our CARIC-DA system. Considering that the two Intel platforms show the same trend for different aspects of the performance, we use the Intel E7 to represent the Intel platforms.

3.4 Effect of Increasing Database Size

In this section, we describe the effectiveness of our proposal for a variety of datasets. Two much larger datasets, containing 1,000,000 tuples (table size 377 MB) and 3,000,000 tuples (table size 1,130 MB) were used in this experiment, whereas the dataset in the previous experiments comprised 100,000 tuples (table size 38 MB). We compared the average response time and cache utilization with 48 concurrent clients for a 24-CDP-based CARIC-DA system with those for the baseline system.

The Intel platform was not sensitive to the size of the dataset (Fig. 10 (a)). The performance changed slightly for both the CARIC-DA system and the baseline system as the dataset increased. As we have mentioned earlier, on the Intel platform the performance is dominated by instruction misses, therefore the performance of the cache levels is not affected by changing the size of the dataset (Fig. 11 (a)).

On the AMD platform, the performance remained stable as we increased the size of the dataset to 377 MB. However, when the size of the dataset was greatly increased to 1,130 MB, the performance decreased for both the baseline system and the CARIC-DA system. Moreover, the performance gap between the CARIC-DA system and the baseline system narrowed (Fig. 10 (b)). Firstly, we consider that the decrease in performance for both systems is reasonable. When the dataset is greatly increased to 1,130 MB, the L2 cache has the potential to access a much larger dataset in both systems and undoubtedly the cache miss rate will increase. We observed that the cache miss rate increased for both the CARIC-DA and baseline systems as the dataset increased (Fig. 11 (b)). In the CARIC-DA system, the L2 cache accesses a smaller subset, whereas in the baseline system, the L2 cache accesses the whole dataset. However, when the size of the whole dataset greatly increases to 1,130 MB, the subset size in the CARIC-DA system is also greatly increased (to 47 MB). The sizes of both the whole dataset and our subset greatly exceeded the capacity of the small L2 cache (512 KB). The size difference between the subset in the CARIC-DA system and the whole dataset in the baseline system becomes less significant when compared with the size of the very small L2 cache, and the difference in performance between the two systems therefore decreases.

For a real-world workload, data access is skewed and the frequently accessed dataset is much smaller than the whole DB. Taking this skew into account, our CARIC-DA system is considerably advantageous over the baseline system when dealing with GB-size datasets.

3.5 Tolerance to Skew

We use skewed datasets of 3,000,000 tuples (table size 1,130 MB), 10,000,000 tuples (table size 3,766 MB), and 15,000,000 tuples (table size 5,649 MB) to evaluate the performance of CARIC-DA when dealing with skewed datasets. The system settings were the same as those used in the previous section. Here, queries access the table in a skewed distribution, and for the skewed data access, 50% of the queries access 10% of the tuples (Zipf (0.75) distribution).

When the uniform data access was changed into the skewed data access, the performance of the baseline system improved slightly, while the performance of the non-load-balanced CARIC-DA system dropped sharply. The re-

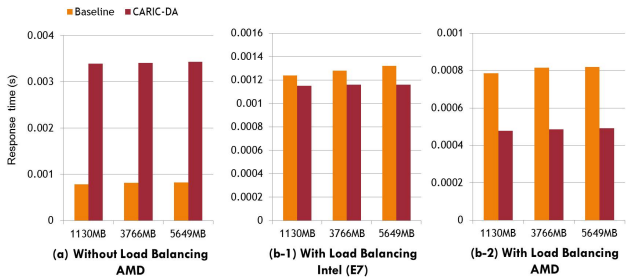

Fig. 12 Performance with skewed data access.

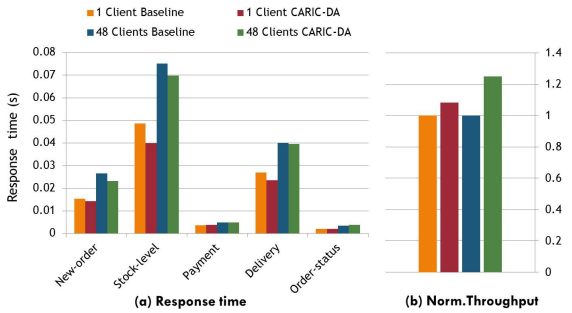

Fig. 13 Performance for the TPC-C benchmark.

sults for the AMD platform are shown in Fig. 12 (a), and the CARIC-DA system was four times slower than the baseline system.

After enabling the dynamic-load-balancing function in the CARIC-DA system, the performance improved dramatically; the CARIC-DA system outperformed the baseline system by 7% on the Intel platform (Fig. 12 (b-1)) and by 39% on the AMD platform (Fig. 12 (b-2)) for the dataset of 1130 MB. For the other two datasets, the performances of the dynamic-load-balanced CARIC-DA system were 9% and 12% above those of the baseline system on the Intel platform and 41% and 40% above those on the AMD platform. These results confirmed the efficiency of the CARIC-DA system when dealing with skewed datasets and also substantiated our claim that skew favors the CARIC-DA system (the advantage increased from 5% (uniform access) to 12% (skewed access) on the Intel platform and from 7.3% (uniform access) to 41% (skewed access) on the AMD platform).

3.6 TPC-C Benchmarking

In this section, we report experiments conducted under the more realistic workloads of the OLTP benchmark TPC-C to compare the performance of our CARIC-DA system with that of the baseline system. We consider it is important for a DBMS system to perform well, not only for simple workloads but also in realistic applications. TPC-C models an OLTP database for a retailer. It consists of five transactions that follow customer orders from the initial creation to the final delivery and payment. We used a 24-warehouse TPC-C dataset (~4.8 GB).

In Fig. 13 (b) we compare the normalized throughput of the baseline and CARIC-DA systems achieved under different concurrency levels on the AMD platform. With the setting of no concurrency (one-client setting), the CARIC-DA system outperforms the baseline system by 10%. The CARIC-DA system can achieve 25% throughput improvement under a high-concurrency setting (48-client setting).

The average response time for the five transactions is shown in Fig. 13 (a). When there are no concurrent transactions, the CARIC-DA system can reduce the execution time of the New-order transaction by 7% and can significantly reduce the execution time of the Stock-level transaction by 18%. The Stock-level transaction retrieves the item information of the most recent 20 orders in a specific district. In our CARIC-DA system only the specific DBP processes the New-order transactions in which the item information of the specific district is also accessed. When the Stock-level transaction runs after the New-order transaction, there is a higher probability that the item information of the recent orders exists in the private cache level. However, for the baseline system, the specific DBP has to deal with the New-order transactions from all the districts, and for the recent order information of the specific district, the probability of a hit for the private cache is relatively low. That is why the Stock-level transaction has better performance in our proposed system. In the high-concurrency experiment, the New-order transaction can be further optimized by 11%. For concurrent transactions, the New-order transactions benefit more from the CARIC-DA system than in the case of no concurrency. Moreover, we observed that the CARICA-DA system can achieve lower abortion rates of New-order transactions than the baseline system. This is the other reason for the 25% increase in throughput with the CARIC-DA system.

4. Related Work

The performance gap between processors and main memory devices has been increasing exponentially for over two decades. For single-core systems, substantial effort has already been made to improve the DBMS performance by solving the problem of large memory-access latency via cache-optimized data structures [25] and algorithms [26]. PAX [5] restructures the data layout in the disk and memory so that servicing one cache miss can "prefetch" other data into the cache to avoid subsequent cache misses. Cache-conscious optimizations for the B+ tree [6] typically set the node size to several cache blocks to increase spatial data locality. With the arrival and popularization of multithreading and multicore processors, new research problems are emerging. Concurrent and parallelized query executions reduce the effectiveness of single-query optimizations, and the data-stall problem is becoming severe and complex.

4.1 Optimizing DBMSs on Multicore Platforms

There was much early work on optimizing the performance of DBMSs for OLTP transactions on simultaneous mul-

tithreaded processors (SMPs) [27] and chip multiprocessors [28]. In recent years, enlarged LLC caches on multicore platforms have been used in an attempt to achieve better performance by capturing larger working sets. Unfortunately, many DB operations have relatively modest primary working sets and cannot benefit from larger LLCs. Furthermore, larger LLC caches require more time to service a hit. This indicates that to focus merely on the LLC is insufficient for attaining maximum performance and sustaining high throughput. Hardavellas *et al.* [9] noticed this problem and pointed out that DB systems must optimize for locality in high-level caches (such as an L1 cache). STEPS [10] improves the L1 instruction-cache performance by scheduling threads in OLTP workloads, but improving the data locality remains a problem for high cache levels. CARIC-DA focuses on the private-cache levels (L1 and L2 caches), which are closer to the processor than the LLC.

DORA [29] is a typical system that focuses on the optimization for OLTP workloads on multicore platforms. Furthermore, it shares similarities with CARIC-DA in terms of range partitioning. In contrast to CARIC-DA, which optimizes the cache performance, DORA is designed to reduce lock contentions. DORA is based on a DBMS that supports multiple threads, and introduces several changes to the lock manager. It may be very difficult for existing DBMSs to benefit from DORA as enabling a conventional thread-to-transaction DBMS to support multiple threads for one transaction is rather complex. On the other hand, CARIC-DA does not employ any functions of existing DBMSs and can be implemented as pure middleware over existing systems.

4.2 OS-Level Optimization

In addition to developing cache-conscious systems involving only DBMSs, some researchers are now trying to achieve this goal via collaboration between the DBMS and the OS. The MCC-DB [7] introduces functions from the OS to improve cache utilization for DBMSs on multicore platforms. This is similar to CARIC-DA in terms of taking advantage of OS capabilities. However, MCC-DB introduces a cache-partitioning function, which is not supported by general-purpose OSs. Therefore, considerable modification to existing systems would be inevitable. In contrast, we rely on the CPU-affinity function, which is well supported by most modern OSs.

The CPU-affinity function has already been used for performance improvement in a variety of applications, although not yet for DBMS applications. Foong *et al.* [30] presented an experiment-based analysis of network-protocol performance under various affinity modes on SMP servers and report good gains in performance by changing only the affinity modes. However, their results are limited to network applications and cannot be directly adopted by DBMS applications, which are much more complex and have intensive data access. Our CARIC-DA framework takes into account data allocation for data-intensive applications and introduces the CPU-affinity function into DBMS applications.

5. Conclusions

With the number of cores increasing and the much more complex cache levels found in modern multicore processors, it is increasingly important to bring data much closer to the processor in addition to improving cache utilization in the LLC. In this paper, we introduced a new approach, CARIC-DA, which is the first work addressing the problems of private caches on multicore platforms for DBMSs. The CARIC-DA system presented in this paper involves an effective collaboration between the DBMS and the OS to improve private-cache utilization on a multicore platform for OLTP-style transactions. Not only can it maintain better locality for the private cache of each core, it can also balance loads dynamically across different cores. An advantage of CARIC-DA is that it is pure middleware, enabling existing DBMSs to be used directly without modification. CARIC-DA is therefore more practical than many existing approaches, which need to modify existing DBMSs to be able to utilize multicore environments well. Experiments show that the CARIC-DA framework can provide both higher performance and scalability for both uniform and skewed datasets. On the AMD multicore platform, the L2 cache miss rate can be reduced greatly (by 56%) for select-intensive transactions and the throughput can be improved by 25% for TPC-C transactions using our CARIC-DA framework.

In future work, improvements and extensions to the existing approach will also be considered. CARIC-DA focuses on optimization for OLTP-style applications, but providing better cache-access patterns for various decision support systems will also be a future challenge.

References

[1] F. Xi, T. Mishima, and H. Yokota, "CARIC-DA: Core affinity with a range index for cache-conscious data access in a multicore environment," DASFAA, pp.282–296, 2014.
[2] F. Xi, T. Mishima, and H. Yokota, "Optimizing concurrent query execution on modern multisocket multicore platform," FTSIS, pp.125–130, 2014.
[3] J. Cieslewicz and K.A. Ross, "Database optimizations for modern hardware," Proc. IEEE, vol.96, no.5, pp.863–878, May 2008.
[4] A. Ailamaki, D.J. DeWitt, M.D. Hill, and D.A. Wood, "DBMSs on a modern processor: Where does time go?," VLDB, pp.266–277, 1999.
[5] A. Ailamaki, D.J. DeWitt, M.D. Hill, and M. Skounakis, "Weaving relations for cache performance," VLDB, pp.169–180, 2001.
[6] J. Rao and K.A. Ross, "Making B+tree cache conscious in main memory," SIDMOD, pp.475–486, 2000.
[7] R. Lee, X. Ding, F. Chen, Q. Lu, and X. Zhang, "MCC-DB: Minimizing cache conflicts in multi-core processors for databases," VLDB, pp.373–384, 2009.
[8] "AMD family 10h server and workstation processor power and thermal data sheet." http://support.amd.com/TechDocs/43374.pdf
[9] N. Hardavellas, I. Pandis, R. Johnson, N.G. Mancheril, A. Ailamaki, and B. Falsafi, "Database servers on chip multiprocessors: Limitations and opportunities," CIDR, pp.79–87, 2007.
[10] S. Harizopoulos and A. Ailamaki, "STEPS towards cache-resident transaction processing," VLDB, pp.660–671, 2004.

[11] R. Johnson, I. Pandis, N. Hardavellas, A. Ailamaki, and B. Falsafi, "Shore-MT: A scalable storage manager for the multicore era," EDBT, pp.24–35, 2009.
[12] "Transaction Processing Performance Council. TPC-C v5.5: Online transaction processing (OLTP) benchmark."
[13] R. Love, "Kernel korner: CPU affinity," Linux Journal, vol.2003, no.111, p.8, July 2003.
[14] T.I. Salomie, I.E. Subasu, J. Giceva, and G. Alonso, "Database engins on multicores, why parallelize when you can distribute?," EuroSys, pp.17–30, 2011.
[15] M. Stonebraker, "Technical perspective-one size fits all: an idea whose time has come and gone," Commun. ACM, vol.51, no.12, p.76, Dec. 2008.
[16] "MongoDB." http://mongodb.com
[17] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's highly available key-value store," SOSP, pp.205–220, 2007.
[18] M. Stonebraker, D. Abadi, D.J. DeWitt, S. Madden, E. Paulson, A. Pavlo, and A. Rasin, "MapReduce and parallel dbmss: Friends or foes?," Commun. ACM, vol.53, no.1, pp.64–71, 2010.
[19] "PostgreSQL: The world's most advanced open source database." http://www.postgresql.org/
[20] "Oprofile: A system profiler for Linux," 2004. http://oprofile.sf.net
[21] "Intel 64 and IA-32 Architectures Software Developers Manual." http://download.intel.com/design/processor/manuals/253668.pdf
[22] "Intel 64 and IA-32 Architectures Optimization Reference Manual." http://www.intel.com/content/dam/doc/manual/64-ia-32-architectures-optimization-manual.pdf
[23] "BIOS and Kernel Developer's Guide for AMD Family 10h Processors," 2010. http://support.amd.com/TechDocs/31116.pdf
[24] D. Levinthal, "Performance analysis guide for Intel Core i7 processor and Intel Xeon 5500 processors," Intel Performance Analysis Guide, 2009.
[25] J. Zhou and K.A. Ross, "Buffering accesses to memory-resident index structures," VLDB, pp.405–416, 2003.
[26] P. Boncz, S. Manegold, and M.L. Kersten, "Database architecture optimized for the new bottleneck: Memory access," VLDB, pp.54–65, 1999.
[27] K. Keeton, D.A. Patterson, Y.Q. He, R.C. Raphael, and W.E. Baker, "Performance characterization of a Quad Pentium Pro SMP using OLTP workloads," ISCA, pp.15–26, 1998.
[28] P. Ranganathan, K. Gharachorloo, S.V. Adve, and L.A. Barroso, "Performance of database workloads on shared-memory systems with out-of-order processors," ASPLOS, pp.307–318, 1998.
[29] I. Pandis, R. Johnson, N. Hardavellas, and A. Ailamaki, "Data-oriented transaction execution," VLDB, pp.928–939, 2010.
[30] A. Foong, J. Fung, and D. Newell, "An in-depth analysis of the impact of processor affinity on network performance," ICON, pp.244–250, 2004.

Fang Xi received her B.E., and M.E. degrees from Dalian University of Technology, China, in 2007 and 2009, respectively. She is currently a Ph.D. student at Tokyo Institute of Technology. She is engaged in research on data engineering and database systems. She is a member of IPSJ.

Takeshi Mishima received his B.E., and M.E. degrees from University of Tsukuba in 1994 and 1996, respectively. He received his Dr.Eng. degree from Tokyo University in 2010. He has been with NTT Corporation since 1996. His research areas include database systems and cloud computing. He is a member of IPSJ and DBSJ.

Haruo Yokota received his B.E., M.E., and Dr.Eng. degrees from Tokyo Institute of Technology in 1980, 1982, and 1991, respectively. He joined Fujitsu Ltd. in 1982, and was a researcher at ICOT for the Japanese 5th Generation Computer Project from 1982 to 1986, and at Fujitsu Laboratories Ltd. from 1986 to 1992. From 1992 to 1998, he was an Associate Professor at Japan Advanced Institute of Science and Technology (JAIST). He is currently a Professor at the Department of Computer Science in Tokyo Institute of Technology. His research interests include the general research areas of data engineering, information storage systems, and dependable computing. He has been a chair of ACM SIGMOD Japan Chapter, a trustee board member of IPSJ and the DBSJ, and the Editor-in-Chief of Journal of Information Processing. He is currently a vice chair of DBSJ, an associate editor of the VLDB Journal, a fellow of IEICE and IPSJ, a senior member of IEEE, and a member of JSAI, ACM, and ACM-SIGMOD.