論文 / 著書情報
Article / Book Information

論題(和文)	マルチコア環境におけるRelational Join に対するコアメモリ割り当の影 響の評価
Title(English)	A performance study of core-memory location effects for relational joins on multi-core computers
著者(和文)	西方, 三島健, 横田治夫
Authors(English)	Fang Xi, Takeshi Mishima, Haruo Yokota
出典(和文)	, , ,
Citation(English)	, , ,
発行日 / Pub. date	2013,

マルチコア環境における Relational Join に対するコアメモリ割り当の影響の評価

西　　方 †　三島　健 ††　横田　治夫†††

† 東京工業大学大学院 情報理工学研究科　〒 152–8552 東京都目黒区大岡山 2-12-1
†† NTT ソフトウェアイノベーションセンタ　〒 180–8585 東京都武蔵野市緑町 3-9-11
††† 東京工業大学大学院 情報理工学研究科　〒 152–8552 東京都目黒区大岡山 2-12-1
E-mail: †xifang@de.cs.titech.ac.jp, ††mishima.takeshi@lab.ntt.co.jp, †††yokota@cs.titech.ac.jp

A performance study of core-memory location effects for relational joins on multi-core computers

Fang XI†, Takeshi MISHIMA††, and Haruo YOKOTA†††

† Department of Computer Science, Tokyo Institute of Technology
Tokyo, 152-8552 Japan,
†† Software Innovation Center NTT Japan
Tokyo, 180–8585 Japan
††† Department of Computer Science, Tokyo Institute of Technology
Tokyo, 152-8552 Japan,
E-mail: †xifang@de.cs.titech.ac.jp, ††mishima.takeshi@lab.ntt.co.jp, †††yokota@cs.titech.ac.jp

Abstract In recent years, the technology of multi-core processor has been improving dramatically. As the number of cores on a chip increases, the memory location relative to cores is beginning to vary the memory access time. In order to take full advantage of the hardware for DBMSs, it is important to clear how the various core-memory locations affect DBMS performance. In this paper, we provide an experimental performance study of the core-memory location effects for different relational joins. We observed that different join operations had different sensitivity to the core-memory locations. Our discussion and insights are useful for DBMS performance optimization on multi-core computers.

Key words joins multi-core computers DBMS

1. Introduction

Driven by the Moore 's Low, computer architecture has entered a new era of multi-core structures. Recently microprocessor manufacturers find that it has become increasingly difficult to make CPUs go faster due to size, complexity, clock skews and heat issues. So they continue the performance curve by another route of developing dual core and multi-core processors. That is, putting multiple CPUs on a single chip and relying on the parallelism ability to get higher performance gain which brings the computing world into so-called " multi-core era ". The multi-core processor is the mainstream now and there will be many-core processors with more cores on one die in the near future [1].

However, these processors are far from realizing their potential performance when dealing with data-intensive applications such as database management systems (DBMSs). This is because the advance in the speed of commodity multi-core processors far outpaces that in memory latency, leading to processors wasting much time waiting for required data item. How to efficiently use the new hardware resources for DBMSs becomes into hot topics. A lot of researchers focus on how to overcome the memory wall by changing the data layout [2], index structures [3], providing cache friendly

algorithms [4] [5] for DBMSs.

The data starve problem becomes severe when the processor manufactures is having more and more cores integrated in a single chip. Therefore the Non-Uniform-Mmory-Access (NUMA) is being used inside a multi-core processor to provide efficient memory accesses and is becoming critical technique as the number of cores on chip keeps increasing in the multi-core processors. The design of NUMA yields to two levels of memory accesses: independent direct local memory accesses and remote memory accesses using a shared interconnect. Because there will be less contention on the shared resources, it offers better scalability. However, one of the most important performance bottlenecks in a NUMA system is an overhead caused by remote memory accesses. Under NUMA, a processor can access its own local memory faster than remote memory, which is local memory to another processor. In order to improve the performance for DBMSs on the multi-core platforms, it is important to make clear how much the remote memory access affects the performance of DBMS queries which usually have very frequent and large quantities of data accesses. In this paper, several experiments are conducted on a NUMA based multi-core platform to clarify how the different core-memory location settings affect the performance of join operations. In our experiment, for an equal join query, we observed the remote memory access has dramatic effect on the nest loop join operations and little effect on the sort merge join operations. The insights achieved in our experiments are useful for further research of join performance optimization on multi-core platforms and are also referenceable for optimization of different applications.

2. Related Work

2.1 NUMA

Modern CPUs operate considerably faster than the main memory they use and the processors increasingly "starved for data", have had to stall while they wait for memory accesses to complete. Multi-processor systems make the problem considerably worse, and the limited memory bandwidth provided the key to achieving high performance for these platforms. NUMA attempts to address this problem by providing separate memory for each processor. In NUMA, main memory is physically distributed, i.e., partition in so-called "locality domains(LDs)", but the main memory is logically shared, meaning that all memory locations can be accessed by all processors in the system transparently. However, since mainly memory is physically distributed, the access bandwidths and latencies may vary, depending on which processor accesses a certain part of memory.

Under the NUMA architucture, there are two levels of memory accesses: independent direct local memory accesses and remote memory accesses using a shared interconnect. Because there will be less contention on the shared resources, it brings better scalability. One of the most important performance bottlenecks in the NUMA system is an overhead caused by remote memory accesses. A processor can access its own local memory faster than non-local memory, that is, local memory to another processor.

There have been many research studies targeting NUMA systems in terms of performance evaluation, analysis, and optimization on clusters and multi-processor systems. Earlier NUMA systems such as HP/Convex Exemplar were investigated in [6]. The performance of cluster with NUMA nodes is investigated for scientific applications [7]. The NUMA effects to the OLTP based DBMS is also analyzed in early research [8].

Recently, with the wide spread utilizations of multi-core processors and the trend of increasing the number of on-chip cores, the NUMA is not only used in multi-processor platforms and clusters but also used in multi-core processors. For the AMD Opteron6174 processor which integrated 12 cores in a single socket, there are two nodes and each node has its own local memory. On this kind of multi-core platform, the access latency to a memory location is different depending on where processor core the program is running. As the NUMA is becoming universal, it is important to make clear how the remote memory access affects the performance of DBMS to achieve better performance by avoiding the remote memory accesses.

In this paper, we mainly focus on the NUMA effects on the join operations which are frequently used operations for OLAP applications. The platform we used includes four multi-core processors and there are two nodes in each chip, which is different from neither the cluster nor multi-processor architectures which used in early researches.

2.2 Join processing

Join is an important database application. As the computer architecutre evolves, the best join algorithm may change. The research of algorithm to maximize the favor of new hardware architectures is never stopped. On multi-core platforms, single-threaded query execution is not promising to meet the high expectations of database users. Only query engines relying on intra-query and intra-operator parallelism will be able to meet the expectations instantaneous response time of operational business intelligence users if large main memory databases are to be explored.

Therefore, a lot of researches focus on the efficient and scalable parallel join operations on multi-core platforms. The well-known radix join algorithm of MonetDB [9] pioneered the new focus on cache locality by repeatedly partitioning

the arguments into ever smaller partitions. An Intel/Oracle team [10] adapted hash join to multi-core CPUs. They also investigated sort-merge join and hypothesized that due to architectural trends of wider SIMD, more cores, and smaller memory bandwidth per core sort-merge join is likely to outperform hash join on upcoming chip multiprocessors. He et al. [11] developed parallel nested-loop, sort-merge, and hash joins on GPUs. The work [12] founded the last level cache (LLC) conflict when different kinds of joins concurrently run together.

Unlike early works which mainly focus on the effects of cache level, TLB and SIMD, we analyze the effect of NUMA on the different join operations because the multi-core based NUMA effect has not been analyzed yet. Furthermore, as more cores and large memory are equipped into the multi-core platforms, considering NUMA is decisive for large scale databases.

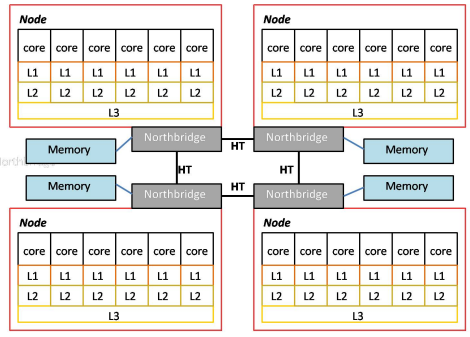
3. Experimental Setup

3.1 Multi-core platform

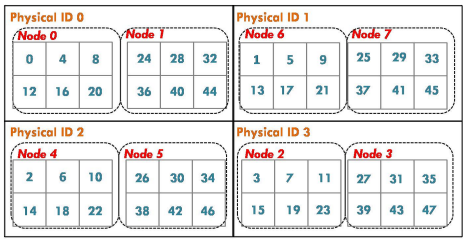
Our evaluation is based on PostgreSQL [15] and Linux (2.6.35-22-server). We set the `shared_buffers` of PostgreSQL to 2GB and the SHMMAX value of the Linux kernel to 30GB in order to support the database settings of `shared_buffes`.

The hardware for the DB server is a 48-core AMD Opteron system [13] [14]. It has four processor sockets, with a 12-core AMD Opteron6174 processor per socket. Each core has a clock speed of 2.2 GHz and had a 128-KB L1 cache (2-way associative 64-KB data cache + 64-KB instruction cache) and a 512-KB L2 cache (16-way associative). Each processor has 12MB L3 cache shared by 12 cores. Each core has a 40-entry translation look aside buffer. The server to evaluate the join queries has 32 GB of off-chip memory and 500GB Hard Disk.

Each processor has 2 nodes, each of which has 6 cores. Each node has its own local memory of 4GB and all memory accesses from the cores in the same node is controlled by the Northbridge of the node. First, the memory request is analized in the Northbridge. If the address to access the memroy is outside the local memory, the Northbridge will transport the requests to the proper node through the HyperTransport (a link for interconnection of node structures)(see Figure 1 (a)) [14]. Each processor has four HyperTransport links which are used to connected to other nodes and our platform has eight nodes. Therefore, all of the nodes are not connected directly. We define two kinds of remote nodes for a particular node: the close remote node which is connected directly with HyperTransport links and the far remote node which is not connected directly with HyperTransport links.



(a) Node and memory relations under NUMA



(b) Logical core and node distribution in our platform

Node	0	1	2	3	4	5	6	7
0	10	16	16	22	16	22	16	22
1	16	10	16	22	22	16	22	16
2	16	16	10	16	16	16	16	22
3	22	22	16	10	16	16	22	16
4	16	22	16	16	10	16	16	16
5	22	16	16	16	16	10	22	22
6	16	22	16	22	16	22	10	16
7	22	16	22	16	16	22	16	10

(c) Node distances

Figure 1 Logical core and node information in our platform

For close remote nodes, we further devide them into two subcategories: neighbor node and close remote node. The two nodes on the same processor are neighbor nodes. The two nodes on different processors, but connected directly with HyperTransport link are defined as close remote node. Because there are not any HyperTransport links between a node and it's far remote nodes, the memory request to the far remote node is first translated to a close remote node, and then is translated to the target far remote node.

The relationship between the nodes can be observed by the command `numactl` and the value of node distance shows access time. For a specific core there are four types of memory locations consistent with different node distances: the local memory, the neighbor memory which belongs to the neighbor node, the close remote memory which belongs to the close remote nodes, the far remote memory which belongs to the far remote nodes. The memory access latency depends on the node distance. The access time to local memory is fast, and the most time-consuming access uses far remote memory. Figure 1 is the relationship of different processor cores with different logical numbers and nodes with different node numbers in our platform (Figure 1 (b)), and the distance between different nodes which is get by the commond of `numactl` (Figure 1 (c)).

3.2 Data set

In our experiments, we use relations of 16 bytes wide tuples (8 bytes join key and 8 bytes payload) and consisting of uniformly distributed unique random numbers for the join key field. In the join experiments, the join hit rate is one, and both of the two tables have 16M tuples.

We execute an equal-join on two tables:

$SELECT\ count(*)$

$FROM\ RjoinS$

$ON\ R.joinkey = S.joinkey$

This query ensures only one output tuple is generated thus we can concentrate on join processing cost only.

4. Experiments

In this section, we evaluate the effects of NUMA on three kinds of join operations: indexed nest loop join, merge join, hash join for single join operation and concurrent join queries separately on our eight node multi-core platform.

First, we bind the query to execute on processor cores on the node 0; Next, we used the Linux command of `numactl` to ensure all of the memory used by the PostgreSQL (mainly shared_buffers and work_mem) is located in the local memory of different nodes (Figure 2). For example, under the setting of the neighbor memory, the DB process (join query) runs on the cores of node 0, the memory used by the PostgreSQL is restricted to the 4GB local memory of node 1. Then we achieved the goal to assign the neighbor mermory location to the join query. With this method, we separately set different memory locations to different join queries by separately setting the memory used by the PostgreSQL in different memory sections.

We did not consider the performance of cache level in our system, as we suppose the cache level is the same under different memory location settings. If we run the join operation on one core which is located in the node2, the query will use the cache resources of L1, L2 cache for the specific core and the LLC on the processor 3. If we run the join operation on one core which is located in the node5, the query will use the cache resources of L1, L2 cache for the specific core and the LLC on the processor 2. The private cache of L1, L2 for each core is with the same size. Meanwhile, the LLC on each processor is also with the same size. So we suppose the cache level is the same under the different memory location settings.

4.1 Indexed nest loop joins

4.1.1 Random data access

In the first experiment we evaluate how the different memory locations affects the performance of indexed nest loop joins. We choose the B+tree index for the relationf of R and do not create any index for relation S. We separately conducted two sub experiments: single query experiment and concurrent query experiment. In the single query experiment, we run only one join query on the core of the specific node. In the concurrent query experiment, we concurrently run several equal-join queries on the cores of the same node. Figure 3 is the execution time of join queries with different memory locations.

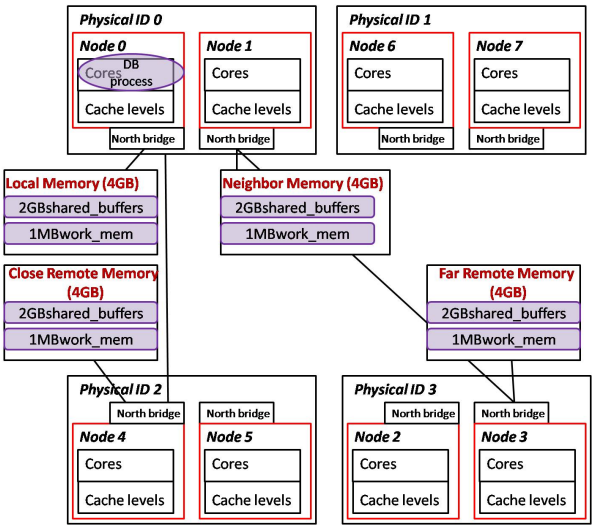


Figure 2 DB process and memory location settings in experiments

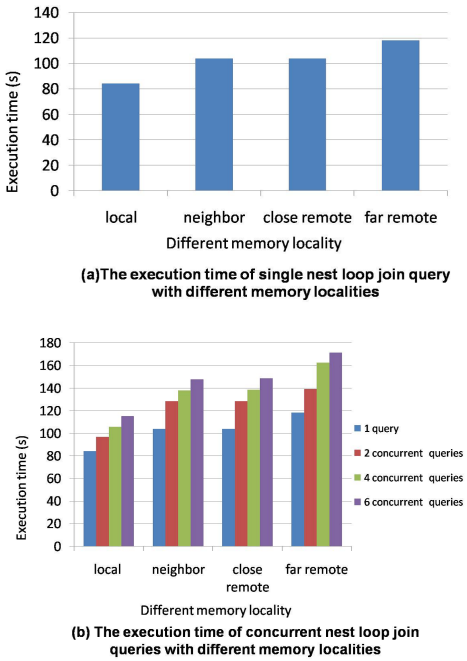


Figure 3 Execution time of nest loop join queries with different memory locations (work_mem=1MB)

From the result of single query execution experiment, we find out different memory locations has dramatic impact on the performance of nest loop joins. The local memory access is the fastest i.e, the query execution time is the shortest. When we set the neighbor memory location to the query, the

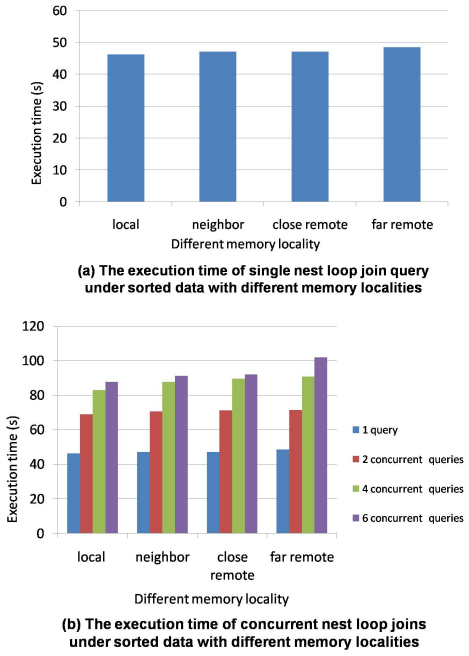


Figure 4 Execution time of nest loop join queries on sorted data with different memory locations (work_mem=1MB)

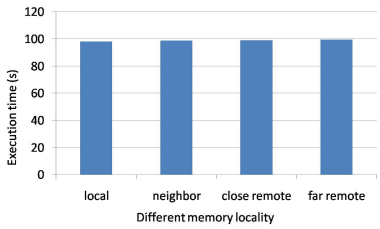


Figure 5 Execution time of sort merge join queries with different memory locations ((work_mem=1MB))

query execution becomes 23% slower compared with running the query on the local memory. The neighbor memory access and the close remote memory access leads to the same performance. That's because, in our platform, the neighbor node(node 1) and the close remote node(node 4) has the same node distance to the local node(node 0). Thus the memory access latency is the same for the cores on the neighbor node and the close remote node. With the setting of far remote memory location, the query performs 40% slower compared with the setting of local memory location. As the node distance between the local node(node 0) and the far remote node(node 3) is the biggest in our platform, the memory access latency for the far remote memory is the highest.

From the concurrent query execution experiment, we observed bigger effect of the memory location on the concurrent nest loop joins shown in the Figure 3(b). X-axis shows the different memory location settings, y-axis shows the query execution time and the different column colors indicate the difference in the number of concurrent queries. With six concurrent queries, the close remote memory access and the far remote memory access brought 28% and 49% increase in execution time respectively, compared with the local memory access. With increasing the number of concurrent queries, it costs more time to finish the join queries. This is because the queries have to share the Last level cache (LLC) when several query concurrently run together. As the size of the LLC is fixed, the more concurrent queries, the serious the cache competition becomes. The increasing of the LLC miss will cause more memory access operations. Thus there are bigger impact of the memory locality on the concurrent joins.

In the previous experiments, we observed dramatic influence of the different memory locations on the performance of the nest loop join operation. And in this experiment, the join operation has a lot of random read operations to the main memory as the tuples of the table have random numbers in the join key field. So we conclude that the random remote memory access latency has significant effects on the performance of the nest loop join operations.

4.1.2 Sequential data access

In the second experiment, we change the data access pattern from ramdom memroy access to sequential accesses. The relation R with the B+tree index is the same with the previous experiment. We initial the relation S with the datas which have sorted order in the join key field and in the previous experiment the data has random order in the join key filed for the relation S. Generally, the random data access and sequential data access have different performances due to the effect of perfetch functions. Figure 4 is the execution time of join queries on sorted data with different memory locations.

Rather than the previous experiment results with random data access for nest loop joins, with the sorted data there is no dramatic impact of different memory locations to the performance of the query. There are very little increase in the query execution time when the query access remote memories compared with access the local memory (2% slower for neighbor, close remote memory locality and 5% slower for close remote memory locality). This is because the effect of the NUMA is mitigated by the prefetcher of the LLC which prefetches data from the memory into the LLC that are likely to be required in the near future. As the memory accesses for the nest loop join on the sorted data are mainly sequential, the required data are mostly cached by the LLC with the prefetcher function.

We also observed that for 2 and 4 concurrent queries, the hardware prefetcher can chieve good performance (in Figure 4(b). The benefit of hardware prefetcher diminished for 6 concurrent queries, and the far remote memory locality increased the query run time by 16% compared with the local

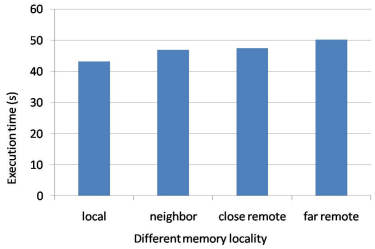


Figure 6 Execution time of sort merge join queries with different memory locations (work_mem=1600MB)

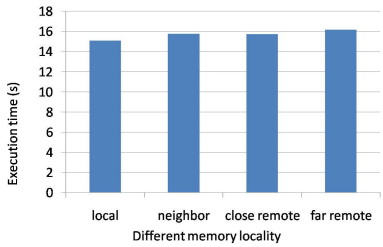


Figure 7 Execution time of sort merge join queries on sorted data with different memory locations (work_mem=1MB)

memory locality). That's because when several queries concurrently run together, different queries may have different data requests and therefore there are random data accesses some times.

4.2 Sort Merge joins

In this section we evaluate the performance of the sort merge join operation under different memory location settings. Figure 5 shows the result for single merge join query. There is little difference with different memory location settings. We then analyze why this join operation is not affected by the NUMA architucture. The merge join is consisting of two sub operations: sort operation and merge operation. We find out the sort operations cannot fit in memory, and some temp results during the sort operation have to be swapped into Hard Disks. We confirm this because the "Sort Method: external merge" in EXPLAIN ANALYZE plans. In this situation, the I/O in the Hard Disk becomes into the main bottleneck which hided the effect of the different memory access latencies under NUMA.

In order to ensure the sort operation can fit in the memory space, we changed the database setting of `work_mem` to 1600MB as the parameter of `work_mem` specifies the amount of memory to be used by internal sort operations. Under this setting, the query plan was changed into "Sort Method: quicksort". Figure 6 shows the results of single merge join query with quick sort operation. We observed that the close remote memory access increased the query execution time by 8.8%, and the far remote memory access increased the query execution time by 16%. Without I/O contention, we can observe the different memory locations exhibit dramatic influence on the sort merge join operations. As there are inevitable random memory read and memory write operations during the in-memory quick sort operation, and the performance of random memory access is affected by different memory location settings.

Beyond the sort operation, as there are mainly sequential data accesses for the merge operation, we suppose the NUMA location will also have little influence to the merge operation. We confirm this assumption by evaluating the sort merge join on two sorted tables. Figure 7 is the execution time for the sort merge join operation on sorted data with different memory locations.

The result proves our assumption. There are very little increase in the query execution time when the query accesses remote memories compared with access the local memory (4% slower for neighbor and close remote memory locality and 7% slower for close remote memory locality). The result is similar with the experiment result of nest loop joins on the sorted data, and the impact of the NUMA is mitigated by the hardware prefetcher.

From the experiment results in the section, we can conclude that the NUMA will not affect the sort merge join dramatically when the sort operation cannot fit in memory. We can observe dramatic influences of the NUMA to the sort merge joins when the in-memory sort can be used.

4.3 Hash joins

In this experiment we evaluate how the different memory locations affect the performance of hash join. We conducted two sub experiments: single query experiment and concurrent query experiment. Figure 8 shows the experiment results.

The remote memory access increases the query execution time for 6%(close remote memory access) and 16%(far remote memory access) compared with the local query access.

For concurrent queries, when the number of concurrent queries becomes 4 or more, a dramatic performance decrease is observed. This is because the hash tables of the concurrent queries cannot be cached in the LLC, and it incurs more LLC misses when several queries concurrently run together. When the hash table cannot be cached in the LLC the NUMA effect is increased into 10%(six concurrent queries with close remote memory access) and 24%(six concurrent queries with far remote memory access).

In the early experiments of hash join, we used the default setting of the `work_mem` (1MB) and we observed the "Buckets: 4096 Batches: 102" in EXPLAIN ANALYZE plans. It indicates that the hash joining tables are too large to fit into memory. In this situation, the database first partitions each table, using a hash algorithm on the values in the join

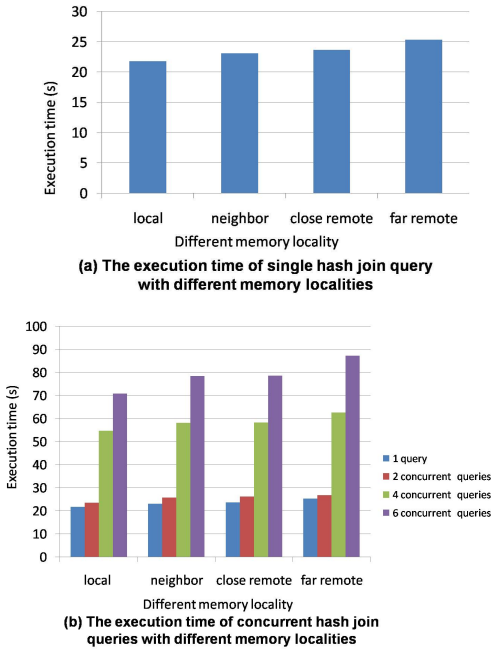


Figure 8 Execution time of hash join queries with different memory locations (work_mem=1MB)

columns. It then goes through each partition in turn, joining the rows from one table with those from the other table that fell in the same partition. Each partition is a Batch.

We increased the `work_mem` to 1600MB and observed "Buckets: 2097152 Batches: 1". Then we redo the single hash join query experiment and Figure 9 shows the experiment results. We observed the remote memory access increased the query execution time for 35%(close remote memory access) and 66%(far remote memory access) compared with the local query access. With the less batches, the hash table becomes larger and the large hash table can not fit into the LLC. It causes a lot of cache miss operations during the hash join operation. Therefore, the query execution time becomes dramatically longer than the multi-batch hash joins as a result of the worse cache performances. With the increase of the cache misses, there will be more frequent memory accesses, and these memory accesses is supposed to be highly random as the number of the Buckets (2097152) is very large. As a result, the differnt memory locations have more dramatic effects on the hash join performance with the setting of big work_mem.

5. Summary and Future Work

Recently, multi-core processors are widely used by many applications and are becoming the standard computing platform. However, these processors are far from realizing their potential performance when dealing with data-intensive applications such as DBMSs. This is because advances in the speed of commodity multi-core processors far outpace

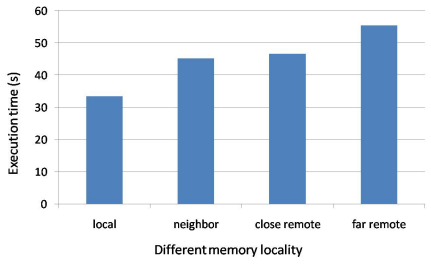


Figure 9 Execution time of hash join queries with different memory locations (work_mem=1600MB)

advances in memory latency, leading to processors wasting much time waiting for required items of data. Thus a lot of work focuses on how to overcome the memory wall through changing data lay out, providing better cache utilizations and so on.

In this paper we analyzed how the NUMA affected the performance of different join operations of PostgreSQL. Under the NUMA architecture, the memory access latency varied, depending on which processor accesses a certain part of memory. We evaluated how the different core and memory loation relationship affected the three kinds of join operations which were nest loop join, sort merge join and hash join on a multi-core platform with three kinds of different memory access latencies. We found out that the remote memory access will greatly decreased the performance of nest loop join operations and increased the execution time by 48% at most when only one relation has B+tree index. We observed little effect on the sort merge join operations especially when the sort operation could not fit in memory. The NUMA architucture increased the running time for the multi-batch hash join by about 10%.

In the future we will focus on the optimizations of join operations on the multi-core platform. We will consider the NUMA effect and reduce the remote memory accesses.

References

[1] Sally Adee "the data: 37 Years of Moore's Law", Proc. of Spectrum, IEEE, pp. 56, 2008.

[2] A.Ailamaki, D.J.DeWitt, M.D.Hill, and M.Skounakis "Weaving Relations for Cache Performance", Proc. of VLDB, pp. 160-180, 2001.

[3] J.Rao and K.A.Ross "Making B+Tree Cache Conscious in Main Memory", Proc. of SIGMOD, pp. 475-486, 2000.

[4] Spyros Blanas, Yinan Li, and Jignesh M. Patel "Design and evaluation of main memory hash join algorithms for multi-core CPUs", Proc. of SIGMOD, pp. 37-48, 2011.

[5] Steven Keith Begley, Zhen He, and Yi-Ping Phoebe Chen

"MCJoin: a memory-constrained join for column-store main-memory databases", Proc. of SIGMOD, pp. 121-1328, 2012.

[6] Brewer.T, and Astfalk.G "The evolution of the HP/Convex Exemplar", Proc. of IEEE, pp. 81-86, 1997.

[7] E.Kornkven, T.El-Ghazawi, and G.Newby "Application performance tuning for clusters with ccNUMA nodes", CSE, pp. 245-252, 2008.

[8] Gerrit Saylor, and Badriddine Khessib "Large scale Itanium2 processor OLTP workload characterization and optimization", DaMoN, 2006.

[9] S.Manegold, P.A.Boncz, and M.L.Kersten "Optimizing Main-Memory Join on Modern Hardware", Proc. of IEEE TKDE, pp. 709-730, 2002.

[10] C.Kim, E.Sedlar, J.Chhugani, T.Kaldewey, A.D.Nguyen, A.D.Blas, V.W.Lee, N.Satish, and P.Dubey "Sort vs. Hash revisited: fast join implementation on modern multi-core CPUs", Proc. of PVLDB, pp. 1378-1389, 2009.

[11] B.He, K.Yang, R.Fang, M.Lu, N.K.Govindaraju, Q.Luo, and P.V.Sander "Relational joins on graphics processor", Proc. of SIGMOD, pp. 511-524, 2008.

[12] Rubao Lee, and Xiaodong Zhang "MCCDB: minimizing cache conflicts in multi-core processors for databases", Proc. of VLDB, pp. 373-384, 2009.

[13] "AMD Family 10h Server and Workstation Processor Power and Thermal Data Sheet", 2010.

[14] "BIOS and Kernel Developer ' Guide(BKDG) for AMD Family 10h Processors ", 2010.

[15] "PostgreSQL: The world's most advanced open source database"