データベースシステム 121－7
情 報 学 基 礎 58－7
（2000．5．25）

分散共有メモリ計算機上における データスキューに対する結合演算の性能解析

中野 美由紀、喜連川 優
{miyuki, kitsure}@tkl.iis.u-tokyo.ac.jp
東京大学生産技術研究所

本報告では、分散共有メモリ計算機上における並列ハッシュ結合演算の処理性能を解析し、データスキューがハッシュ結合演算の処理性能に与える影響について検討した。分散共有メモリ計算機上でバッファ管理方式のコスト式を提案し、そのコスト式に基づき、キャッシュサイズ、データ分布、ノード数などが変化した場合のハッシュ結合演算の性能を詳細に解析した。その結果、我々が提案するバッファ管理方式では、分散共有メモリ計算機のスケラービリティをより引き出すと同時に、データ分布の偏りによる影響が少ないことを確認した。

Effect of Data Skew on Join Execution in Distributed Shared Memory Parallel Machines

Miyuki Nakano and Masaru Kitsuregawa
Institute of Industrial Science, University of Tokyo
7-22-1 Roppongi, Minato-ku, Tokyo, 106 Japan
e-mail : {miyuki,kitsure}@tkl.iis.u-tokyo.ac.jp

In this paper, we analyze the performance of parallel hash joins in DSM machine and consider the data skew impact on hash join execution in DSM machines. Although we have already reported performance evaluation of parallel hash join processing on the DSM architecture, our previous measurements were done by using uniform data distribution and the system resources, such as cache size or number of nodes, were fixed since an actual DSM machine was used. We analyze the processing cost of the proposed four buffer management strategies in DSM machines and report simulation results by varying data distribution, cache size, number of nodes, and so on.

1 Introduction

Recently, with the rapid evolution of information infrastructures such as the Internet and the growth of our information society, large amounts of data are accumulated each day. As a result, many large database systems such as multimedia databases and decision support systems(DSS) were developed. These databases are required to provide quick response to complex queries along with providing facilities to allow for rapid data growth. On the otherhand, many parallel machines have been developed from high-performance microprocessors, utilizing low cost memories whose capacity are very large, possessing a high-bandwidth network and many small low cost disks today. From the background described above, there are many parallel database systems implemented on commercial parallel computers and the TPC-D benchmark results on these parallel systems, such as NCR Teradata, IBM DB2 on SP-2 and Oracle on SUN enterprise series, are reported.

Although the distributed shared memory (DSM) architecture is focused on in parallel processing research now, there is little research on parallel database processing in the DSM environment[4, 2, 6]. Moreover, almost all performance evaluations of the DSM architecture[3, 5] have been done by assuming scientific applications in which access locality is high, while there are no reports analyzing such data intensive applications such as relational DBMS, DSS, multimedia DBMS, in which a large memory space is accessed randomly. So, the research of parallel database processing on the DSM architecture is an important subject to resolve the problem of growing database size and the varied requirements of different user applications.

In [7], we have already proposed the four buffer management strategy for hash join operation and implemented them on an actual DSM Machine HP Exemplar SPP 1600. Then, we have shown that the strategy of considering node locality can achieve substantial performance improvement. However, our previous measurements was done by using uniform data distribution and the system resources, such as cache size, remote memory access cost or number of nodes, were fixed. So,

in this paper, we analyze performance of the parallel hash join in DSM machines employing CC-NUMA by varying data distribution and the system resource parameters. First, we describe an overview of CC-NUMA model and consider memory access characteristics based on the HP Exemplar SPP 1600. Then, we introduce the data buffer allocation and access strategy of parallel hash join and consider the memory access cost of our four buffer management strategies. The simulation results of our strategies are measured by varying cache size, node size, access cost of remote memory and data distribution. From these results, it is shown that DSM architecture is a promising platform for parallel DBMS. Moreover, we show that the buffer management strategy of considering node locality is efficient in improving the performance of parallel database processing in a DSM architecture even when data skew exists.

2 DSM Parallel Computer Architecture

In general, the DSM architecture means that memory allocated on physically distributed memory in each node can be accessed as one virtual memory space by memory coherency mechanisms supported by software or hardware. DSM architecture are also classified by its memory mechanism into COMA(Cache Only Memory Architecture), which regards all memory as a large cache and does not mapped virtual space into specific physical local memory on the nodes, and CC-NUMA(Cache Coherent Nonuniform Memory Architecture), which allocates the virtual space to each distributed physical memory and uses a cache space for maintaining memory coherency. Almost all DSM machine products, such as the HP Exemplar series, SEQUENT NUMA-Q, SGI-Cray Origin 2000 and Data General Aviion NUMA Server, are equipped with CC-NUMA.

We show an overview of a CC-NUMA machine in Figure 1. A CC-NUMA machine consists of nodes connected by an iterconnection network whicn can provide high band width and support a global virtual shared memory by allowing processors in any node to the memories in other nodes. Each node contains a number of processors, a shared memory and a number of disks. In general, a CC-NUMA machine provides the following three memory classes which can be specified when users compile their sources.

1. global memory : is allocated among all nodes equally using a page unit. It can be accessed from all nodes. However, users cannot specify where a page is allocated. This is equal to the shared memory in a shared-everything architecture.

2. local memory : is allocated to the specified node explicitly. So, users can access node local memory with intent to improve access cost. Also, it can be accessed from other nodes.

3. network cache : is used for accessing memory space allocated to the physical memory of the other nodes. It is controlled by network, and memory coherency mechanisms and users cannot specify what data is explicitly cached. However, it is guaranted that the latest access data which is located in a remote node is encached.

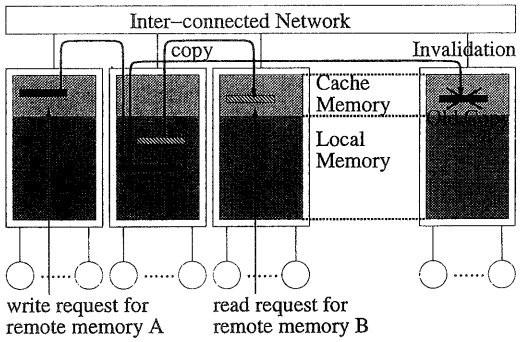

Figure 1: Overview of a CC-NUMA machine

3 Parallel Hash Join Processing on DSM Machine

In this section,we consider buffer access management of the parallel hash join on DSM machines based on the buffer model on shared-everything architecture. In the following, we take an equi-join of relation R and S as follows.

```
SELECT * FROM R,S WHERE R.joinkey = S.joinkey
```

First, we describe the model of parallel hash join processing on the DSM architecture and classify the buffer allocation and access strategy. Then, we introduce the four buffer management strategies, Shared Everything strategy(SE), Shared Hash Table strategy(SHT), Local Hash Table strategy(LHT) and Local Hash Table with Remote Reference strategy(LHT-R).

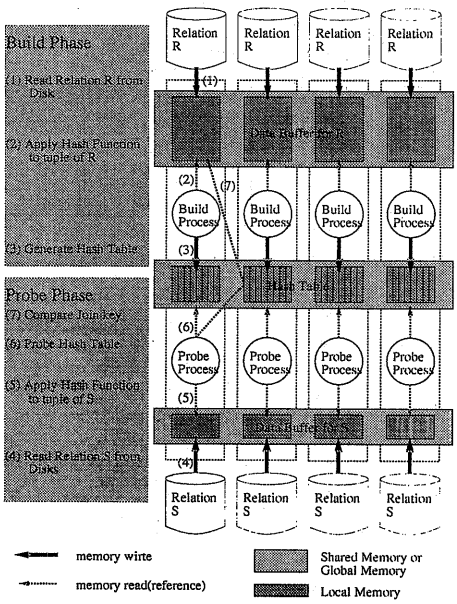

Figure 2: Parallel Hash Join Model on DSM Environment

		SE	SHT	LHT	LHT-R
	Data Buffer	Global	Local	Local	Local
	Hash Table	Global	Global	Local	Local
I/O	Build Phase(1) Data Read	inter node	intra node	intra node	intra node
Build	Build Phase(2) Data Reference	inter node	intra node	inter node	inter node
Build	Build Phase(3) Hash Table Generation	inter node	inter node	intra node	intra node
I/O	Probe Phase(4) Data Read	inter node	intra node	intra node	intra node
Probe	Probe Phase(5) Data Reference	inter node	intra node	inter node	intra node
Probe	Probe Phase(6) Hash Table Reference	inter node	inter node	intra node	inter node
Probe	Probe Phase(7) Build Data Reference	inter node	inter node	intra node	inter node

Table 1: Four Buffer Management Strategies

Figure 2 shows the model of parallel hash join for the DSM environment. A hash join algorithm consists of two phases : the build phase and the probe phase. During the build phase, relation R is partitioned by a hash function and the hash table is generated. The probe phase starts after the hash table is completely generated. Relation S is read from the disks and, for each tuple, the hash tables are probed respectively. The Build Process is responsible for the build phase and Probe Phase is responsible for the probe phase, respectively. Although I/O process is responsible for disk access, it is omitted from Figure 2 for simplicity.

During the build phase, the following steps are repeated until each processor has read its portion of relation R.

B–1 each processor reads relation R from its local disk.
B–2 divides it into buckets using a hash function.
B–3 generates a hash table.

At the probe phase, the following steps are repeated until all tuples of relation S are processed.

P–4 each processor reads relation S from disk.
P–5 divides it into buckets using a hash function.
P–6 probes its hash table
P–7 compares against the join key of relation S with that of relation R. When tuple matching succeeds, a join is performed.

As shown in Figure 2, the hash table, build data buffer and probe data buffer are mainly accessed space in hash join processing and I/O, build and probe processes access to these buffer independently. Although the access cost for these buffers are equal in the shared-everything architecture, as described before, the access cost for these buffers on DSM machines varies depending on where the data buffer is mapped. In the following subsection, we explain the buffer allocation policy and access strategy from the DSM architecture point of view using a model which consists of three buffers and three processes.

As described before, users can allocate the specific memory space onto physical local memory of each node in a DSM machine. So, we can allocate the build and probe data buffers in local memory or in global memory. Similarly, the hash table can be mapped onto local memory or global memory. On the other hand, from the point of view of the three processes, the type of access issued by the processes are classified into local access intra node or global access including remote access between nodes.

As based on the data flow in Figure 2, we consider the access strategy shown in Table 1.

Regarding the build and probe data buffer, I/O process writes these data buffer during the Build

Phase(1) and Probe Phase(4) in Figure 2 after reading data from disks. The build and probe processes read the data buffer during Build Phase(2), Probe Phase(5) and Probe Phase(7). These days, disk are attached to each nodes directly for almost all DSM machines. Moreover, the I/O process issues only write accesses to the buffer. So, since write cost is higher than read cost, the data buffer for build and probe relations should be allocated on local memory explicitly. As for the hash table area, it can be allocated to the local memory on each node as an implementation of shared-nothing architecture does. Thus, three buffer allocation policies are considered from the combination of data buffer and hash table allocation as shown in Table 1.

Once the data buffer allocation policy is determined, the access strategy is also fixed by considering data flow and access efficiency as shown in Table 1. For example, consider Build Phase(2) with the build relation buffer in local memory. Two access strategies are given for candidates : the case that I/O process writes data buffer on local memory and build phase on the other node reads it and the case that I/O process writes data buffer on the other nodes and build phase reads it. As discussed above, the cost for the later case is very high, since write costs for remote memory and read cost for data on local memory which is encached on the other node is high as shown in Table 2.

4 Simulation Results

In this section, we analyze the performance of parallel database processing in a DSM machine by using simulation based on the cost formula of the four buffer management discussed above and varying the system resource parameters.

We have reported the measurement results on the HP SPP 1600 of the four strategy in [7]. However, in our previous measurements, the system parameter values such as cache size, node size and memory access cost were fixed since an actual DSM machine was used. Moreover, we used only the uniform data distribution. Thus, in this measurement, we change resource parameters of DSM Machines, such as cache size, number of nodes and access cost of remote memory. The effect of data skew on parallel hash join is also discussed.

4.1 Simulation Environment and Parameters

In this simulation, we assume that our platform

Mapping	Access	Invalidation	Fault	Lock	cost(usec)
Local	read	No	No	No	0.8
Local	read	Yes(*)	No	No	5.1
Local	write	No	No	No	0.8
Local	write	Yes(**)	No	No	5.8
Local	write	No	Yes	No	5.1
Local	write	No	No	Yes	6.0
Local	write	Yes	No	Yes	16.8

*:invalidated, **:invalidate

Mapping	Access	Invalidation	Fault	Lock	Cache hit	cost (usec)
Remote	read	No	No	No	Yes	0.8
Remote	read	No	No	No	No	4.0
Remote	read	Yes	No	No	No	4.6
Remote	write	No	No	No	Yes	2.0
Remote	write	No	No	No	No	4.0
Remote	write	Yes	No	No	No	5.8
Remote	write	No	Yes	No	No	4.6
Remote	write	No	No	Yes	Yes	12.2
Remote	write	No	No	Yes	No	14.5

Table 2: Memory Access Latency of SPP 1600

is a CC-NUMA machine illustrated in Figure 1 and the cost indicated in Table 2 is used as the basic memory cost parameter values. The size of the relation is varied from 8 to 640 MB and the relations are initially uniformly partitioned among the disks. The data distribution is also varied as discussed later. The size of a page is 4KB and a tuple is 64 bytes in length.

In this measurement, we assume that the directory map of all cache lines are already generated and global and local memory is randomly touched in order to equalize access time from each node. Since the data is encached to the specified node, the access time is very different between the node and the other. Moreover, assuming a database server, almost all of the memory space will be randomly accessed after sufficient time passes. In the following results, we omitted the disk access time to clarify the difference of memory access cost among the four strategies.

4.2 Cost Formula Validation on the HP Exemplar SPP 1600

Before presenting our simulation results with various parameters, we compare the simulation results using uniform data distribution to the results measured on SPP 1600 in order to verify our simulation results. The configuration of SPP 1600 is 4 nodes, each node has 8 CPUs(PA-RISC 7200, 120 MHz), 512MB local memory, which are connected

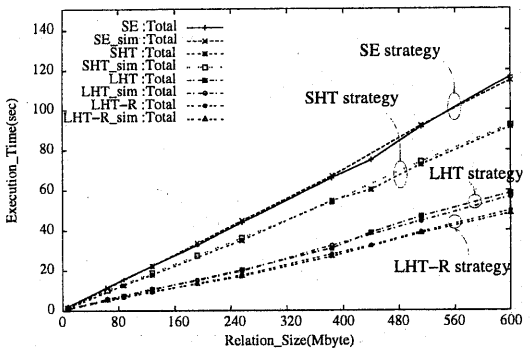

Figure 3: Validation of Simulation Results with Experimental results

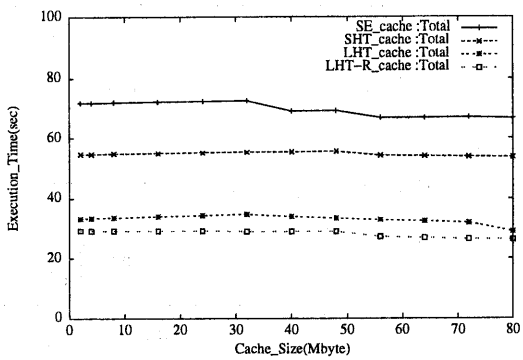

Figure 4: Effect of Cache Size : Total Time

by a crossbar switch, and high bandwidth network employing SCI protocol among nodes which provides distributed shared memory mechanism and maintains memory coherency by 64 byte cache line unit. So, in total, we have 32 processors and 2GB memory.

Figure 3 shows the total execution time for the four strategies calculated by simulation and measured on SPP1600. From this figure, we can see that the performance difference between simulation and experimental results are slightly small.

As shown in Figure 3, the performance of the SE strategy is the worst since all buffers are allocated from global memory. In contrast, the LHT-R strategy shows the best performance from Figure 3. Its performance is improved by 58% compared to the SE strategy. Also, we observe that the performance of LHT and LHT-R strategies which allocates hash table to local memory is better than those of the SE and SHT strategies in which the hash table is allocated from global memory.

In the following subsection, we use the simulation results only.

4.3 Effect of Cache Size

We expect that performance of parallel join execution can be improved when the cache size becomes large, since the memory access cost decreases once the data is encached. Figure 4 shows the simulation results of each strategy by varying cache size. In this measurement, the total relation size is 384 MB, that is, 96 MB per node and the data distribution is uniform. From this figure, we can observe that the execution time for the build phase of each strategy is always constant when cache size increases. This is because the cost for remote memory write is dominant at the build phase of all strategies. In contrast to the build phase, the execution time for the probe phase decreases when cache size increases. With regard to the SE and SHT strategies, since the hash table on global memory is often referenced at the probe phase, almost all remote memory read for hash table becomes cache hit. Moreover, the build relation hold on global memory is also encached on the other node. With regard to the LHT and LHT-R strategies, the improvement of the execution time for the probe phase is smaller than that for the SE and SHT strategies. As for the LHT-R strategy, although the hash table is held on local memory, the probe process also references the hash table located on the other nodes. So, when the cache size becomes larger, the cache hit ratio of the hash table becomes high. However, the part of the build relation is clustered into each node at the build phase, the cache hit ratio of the necessary build data is relatively low. In the LHT strategy, since the probe process does not reference to the remote node memory while searching the hash table, the performance gain is small when the cache size becomes large.

4.4 Effect of Number of Nodes

We show the execution time for each strategy with increasing the number of nodes from 4 to 96 in Figure 5. In this measurement, the cache size is 64MB and the relation size per node is 160 MB. So, when the number of nodes increases, the total relation size is also increase. From these Figures, we can observe that the execution time is almost constant when the number of nodes is larger than 8. This means that the system scalability can be obtained easily in DSM architecture. However, when the number of nodes is varied from 4 to 8,

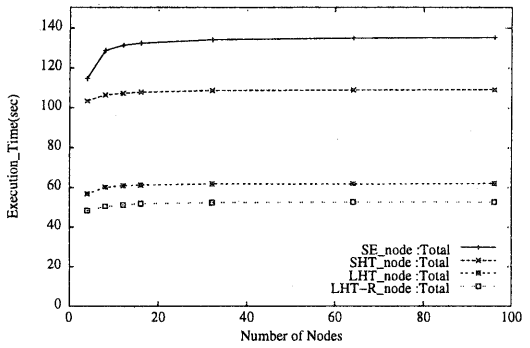

Figure 5: Effect of Number of Nodes : Total Time

each execution time of four strategies increases as shown in Figure 5. As described in the previous subsection, the cache size affects the execution time. So, the performance difference between in the case of 4 nodes and 8 nodes is caused by the cache size. Then, when the number of nodes becomes large, the effect of cache is eliminated. Thus, we find that although DSM architecture provides the system scalability, the system performance is saturated by remote memory access cost when the number of nodes becomes large.

4.5 Effect of Remote Memory Access Cost

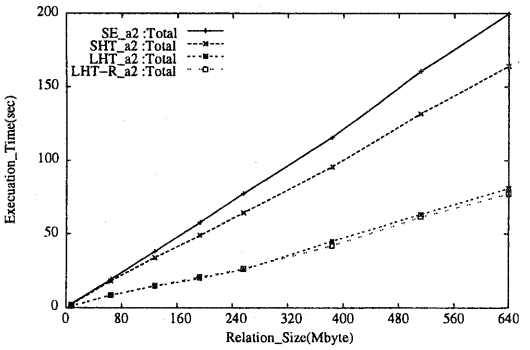

Figure 6: Effect of Remote Memory Access Cost : twice

As observed above, the cost of remote memory access is the dominant factor in the performance of DSM machines. Although the cache size affects the system performance when the relation size is relatively small, almost all execution cost is remote memory access. So, we consider the cost

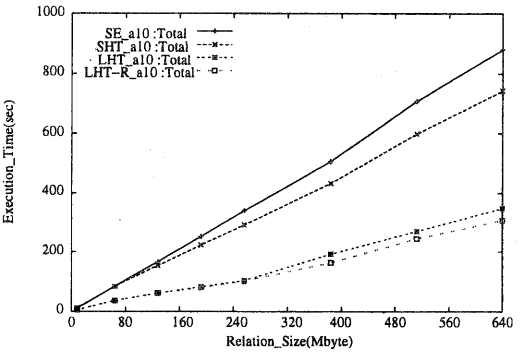

Figure 7: Effect of Remote Memory Access Cost : 10 times

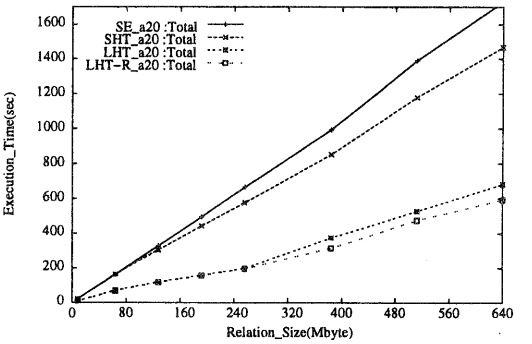

Figure 8: Effect of Remote Memory Access Cost : 20 times

ratio of local memory access to remote memory access. We take the memory cost of the SPP 1600 shown in Table 2 as the base cost ratio of local memory access to remote memory access. Then, the cost ratio is changed twice, 10 times and 20 times. In the case of the cost ratio 20 times, the ratio of local memory access to remote memory access is almost similar to that of DSM software [1].

Figure 6, 7 and 8 shows the execution time of each strategies when the cost ratio is twice, 10 times and 20 times, respectively. The relation size is varied from 16 MB to 640 MB and the cache size is 64 MB. The data distribution is uniform. From these figure, we can observe that the performance difference between SE and LHT(or LHT-R) becomes large when the access cost ratio becomes large.

4.6 Effect of Data Skew

In this subsection, we show the simulation results in existence of data skew. We employ the same data distribution for relation R and S and any load-balancing mechanism is not considered in order to clarify the tolerance to data skew in a DSM machine.

The amount of data skew, *load N%*, is indicated as a fraction of the total relation size which is allocated to one node. The rest of relation is divided evenly amongst all the other nodes. This data skew distribution is also used in [6].

4.6.1 Effect of Data Skew in LHT Strategy

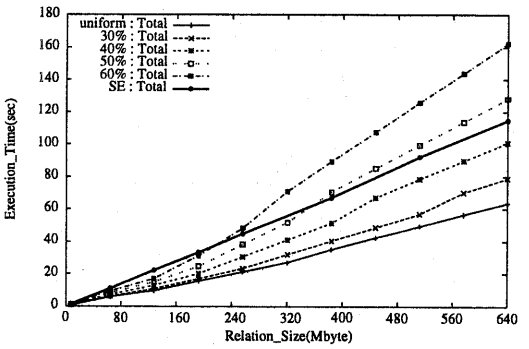

Figure 9: Effect of Data Skew : Total Execution Time of LHT strategy

We show the execution time of LHT strategy by varying the data skew from uniform(load 25%) to *load* 60% in Figure 9. The relation size is varied from 16MB to 640MB and the cache size is fixed at 64 MB. For reference, the execution time of SE strategy is plotted.

In the SE and SHT strategy, the data skew does not affect its performance largely, since any node can reference the whole hash table and the data buffer to be referenced is allocated evenly to the whole nodes. On the other hand, in the LHT strategy, the part of data whose hash value is the same is allocated to the same node which process them. So, the data skew affects the performance of LHT strategy largely as the data skew impacts on the performance of parallel hash join processing in shared-nothing architecture.

As shown in Figures 9, the execution time for the LHT strategy increases when the data skew becomes large. At the build phase, the execution time for the LHT strategy is smaller than that for the SE strategy when the skewed data can be hold on the local memory and the cache. However, when the data to be processed in the node overflows the local memory and the cache size, the execution time increases moderately since the cost of remote memory access increases. At the probe phase, the execution time for the LHT strategy becomes larger than that for the SE strategy, since the cost of remote access, especially local write access with invalidation to the cache of the other nodes, increases as described in section 2. Thus, the total execution time for the LHT strategy becomes large when the data skew is high.

4.6.2 Effect of Data Skew in LHT-R Strategy

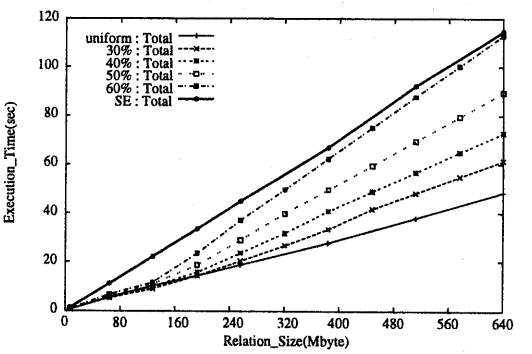

Figure 10: Effect of Data Skew : Total Execution Time of LHT-R strategy

We show the execution time of the LHT-R strategy by varying the data skew from uniform(load 25%) to *load* 60% in Figure 9. The relation size is varied from 16MB to 640MB and the cache size is fixed at 64 MB. For reference, the execution time of SE strategy is plotted.

As shown in Figures 10, the execution time for the LHT-R strategy also increases as well as that for the LHT strategy when the data skew becomes large. Since, as described in section 2, the build phase of LHT-R strategy is the same as the LHT strategy, the execution time for the build phase increases when the skewed data cannot be held on the local memory and the cache. In contrast to the build phase, we can observe that the execution time for the probe phase of the LHT-R strategy is not affected by data skew. This is because the remote or local memory access with invalidation does not happen at the probe phase of LHT-R strategy.

4.6.3 Effect of Data Skew with Varying Memory Access Cost

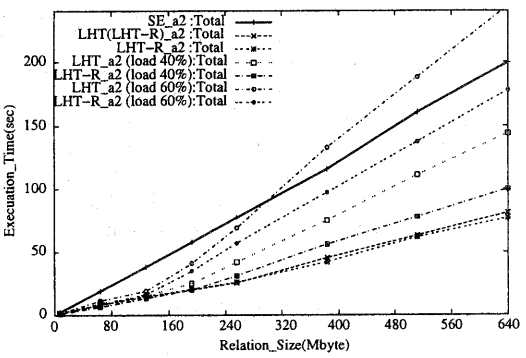

Figure 11: Effect of Data Skew with Varying Remote Memory Access Cost(twice)

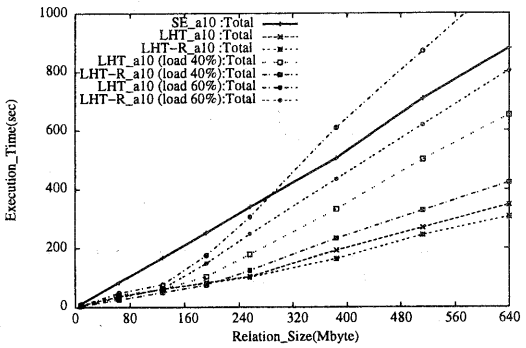

Figure 12: Effect of Data Skew with Varying Remote Memory Access Cost(10 times)

As observed above, both the cost of remote memory access and the data skew affects the performance of parallel join processing in a DSM machine. When the data skew changes, the amount of remote memory access cost also varies in LHT and LHT-R strategies. So, we also study the performance of LHT and LHT-R strategy by varying the data skew and the cost ratio of local memory access to remote memory access.

Figure 11 and 12 shows the execution time of each strategies when the cost ratio is twice and 10 times respectively. For reference, the execution time for the SE strategy is plotted. From these Figures, we can observe that, in comparison with the results of Figure 9 and 10, , both the execution time for the LHT and LHT-R strategy increase moderately. Especially, the performance of the LHT-R strategy becomes better than that of SE strategy even when the data skew is *load*60%. Its performance is improved by 10% compared to the SE strategy in the case that the access cost ratio is 10 times and data skew is 60 %. This means that even when the cost of remote memory access is substantially high, the strategy considering node locality is also important.

5 Conclusions

In this paper, we discuss the performance of the four strategies for parallel hash join processing on DSM machines by using simulator, since the results of previous work was limited since the system resources are fixed. The simulation results of our strategies are measured by varying cache size, node size, access cost of remote memory and data distribution. From measurement results, it is shown that DSM architecture is a promising platform for parallel DBMS since the system scalability comes up to our expectations. Moreover, we can show that the LHT-R strategy shows better performance in comparison to the other strategies even when the data skew exists. Although the performance of SE strategy is slightly affected by the data skew, the execution time for the LHT-R strategy is better than that of the SE strategy whether the data skew exists or not. That is, it is efficient for parallel database processing on DSM architecture to investigate memory access characteristics and choose the optimal access strategy adopting these characteristics.

References

[1] C.Amza, et al.: TreadMarks:Shared memory Computing on Networks of Wrokstations, IEEE Computer, Vol.29, No.2, pp.18-28,1996

[2] L.Bouganim, et al.: Dynamic Load Balancing in Hierarchical Parallel Database Systems, Proc. of 96 VLDB, pp.436-447(1996)

[3] J.B.Carter, et al.: Implementation and Performance of Munin, Proc. of the 13th ACM Sym. on OS, pp152-164, 1991

[4] A.Shardal and J.F.Naughton: Using Shared Virtual Memory for Parallel Join Processing, Proc. of SIGMOD '93, pp.119-128, 1993

[5] J.Chapin, et al.: Memory System Performance of UNIX on CC-NUMA Multiprocessors, SIGMETRICS'95, 1995

[6] S.Pramanik and W.R.Tout : The NUMA with Clusters of Processors for Parallel Join, Int. Journal on Knowledge and Data Engineering, Vo.9, No.4, pp.653-660(1998)

[7] M.Nakano, H.Inai and M.Kitsuregawa : Performance Analysis of Parallel Hash Join Algorithms on a Distributed Shared Memory Proc. of DE, pp.76-85(1998)