計算機アーキテクチャ　113−19
（１９９５．８．２４）

並列データベースシステムにおける多重結合演算処理の最適化とその評価

中野美由紀, 新谷隆彦, 喜連川優
miyuki@tkl.iis.u-tokyo.ac.jp
東京大学生産技術研究所
港区六本木 7-22-1

本報告では, 非共有型並列計算機における多重結合演算処理の最適化方式として, ネットワークコスト, メモリなどのシステム資源の利用効率を考慮した方式を提案する. 本方式では, ネットワークを最大限に有効利用するような部分木を生成し, 生成された部分木を組み合わせることで最終スケジューリング木を得る. 非共有型並列計算機における多重結合演算処理のコスト式を導入し, 部分木の生成するための条件について詳細に述べる. さらに, 従来の最適化方式である left-deep 木,right-deep 木,segmented right-deep 木との性能比較を行ない, われわれの提案する最適化方式が最適な木を生成する率が高いのみならず, 検索空間も従来のフルサーチ方式と比較して十分に実用に適するだけ小さいことを示す.

Optimization of Multi-Way Join in Parallel Database Systems and its Performance Evaluation

Miyuki NAKANO, Takahiko SHINTANI and Masaru KITSUREGAWA
Institute of Industrial Science, University of Tokyo
7-22-1 Roppongi, Minato-ku, Tokyo, 106 Japan

In this paper, we consider parallel multi-way join processing in a shared-nothing environment and propose a new multi-join scheduling algorithm which optimizes a query tree taking into account a finite bandwidth network. Our algorithm first generates sub-tree seeds which fully consumes the network bandwidth in pipeline processing. Then, these sub-tree seeds are combined to produce an optimal query tree. The restriction conditions for generating a sub-tree seed which balances I/O accesses and network transfers and which uses less memory are described in detail. The proposed algorithm for generating the multi-way join plan is evaluated in comparison with former work by using the introduced cost formula. From the evaluation results, not only is the quality of the proposed method better than previously presented algorithms such as left-deep, right-deep and segmented right-deep trees, but the quality of our algorithm does not deteriorate comparatively.

1 Introduction

Recently, there have been a number of commercial database systems implemented on parallel machines. These machines are developed from high-performance microprocessors, utilize low cost memory whose capacity is very large, possess a high-bandwidth network and many small low cost disks. However, the database systems built on these machines are very simplistic from the point of view of parallelization and few of them consider inter-operation parallelism, especially for the multi-way join. So, in order to exploit inter-operation parallelism, significant effort[1, 5, 7, 8, 10, 11, 12, 13] has been focused on developing an efficient multi-way join processing scheduler for parallel database systems. Although several multi-way join optimization methods have already been proposed, almost all of them are targeted for a shared-everything environment and there are few algorithms for shared-nothing systems which consider system resource consumption for network transfers and I/O accesses.

As described in [8], because of the evolution of computer technology, CPU power and memory size continue to increase. Thus, more relations can be loaded into memory and it is necessary to explore a new parallel processing algorithm for multi-way join processing in order to materialize "inter-operation parallelism". Although [9] proposed using "dynamic programming" for the optimization of multi-way joins, this algorithm determines a sequence of multiple join operations and does not consider parallel processing of the multi-way join. In [8], they reported that methods which exploit inter-operation parallelism can achieve better performance than previous methods which processes join operations sequentially. However, their main concern is to demonstrate the importance of inter-operation parallelism in a shared-nothing environment and optimization of the multi-way join is not considered. In [4], the optimization of parallel query processing is discussed, however they only consider CPU-bound and I/O bound tasks executing in parallel in a shared-everything environment. In [1, 7], they propose a greedy heuristic method for multi-way join optimization, however they are only concerned with memory size and source relation size when they make a hash table, and the temporary relation generated after processing some join operations in parallel is always written out to disk. Moreover, the proposed algorithms use a shared-disk environment. [5, 10, 12] also provide optimization methods for the multi-way join and show that performance can be improved by fully utilizing main memory and reducing the write cost of temporary relations. Although the algorithm proposed in [12] is targeted to both shared-everything and shared-nothing environments, it is not clear from [12] how they manage to balance system resource utilization, especially overlapping of network transfers and I/O accesses in a shared-nothing environment. As for parallel dynamic programming in [10], they only consider the cost function of a multi-way join in a shared-everything environment. In [11], the cost formula for multi-join queries are considered in detail, but their research platform is a shared-everything architecture.

In this paper, we propose a new multi-way join plan generation strategy which takes not only memory and CPU consumption but also network bandwidth into account. In the proposed algorithm, the concept of a "balanced seed-tree" is introduced as the unit of parallel processing for the multi-way join. For processing on n-way multi-join in a pipeline manner, n data streams flow through the interconnection network. Although communication bandwidth is increasing rapidly these days, the advent of disk arrays has increased the bandwidth of I/O data streams. This means the network could easily saturate when there are more than a few join operations. In order to fully consume the network bandwidth during pipeline processing, a candidate set of balanced short-trees are generated by introducing some restriction conditions which consider system resource consumption in shared-nothing systems, especially utilization of memory and balancing I/O accesses and network transfers by using a derived cost formula. These balanced seed-trees are combined into the final scheduling tree. The restriction conditions are determined according to some heuristics in order to exploit the harmonious utilization of I/O accesses and network transfers. In our algorithm, maintaining the balance of system resource consumption takes precedence over memory utilization at the pipeline processing level, so that total performance is better than that of the previous methods which only take into account memory utilization.

The remainder of this paper is organized as follows: In section 2, we propose a new algorithm for multi-way join processing and describes this algorithm in detail. The cost formula which considers computer resources such as CPU processing, I/O accesses and network transfers costs is introduced. Then, the generation strategies of a balanced seed-tree are described in the light of bounded system resources. In section 3, we report simulation results by using the cost formula introduced in section 2. The quality of our algorithm is substantially superior in comparison with results of previous work. We conclude this paper and discuss future plans in the last section.

2 New Optimization Algorithm for Multi-Way Join

We propose a new algorithm for multi-way join scheduling algorithm which considers system resources while optimizing a query.

Although communication bandwidth is increasing rapidly these days, the advent of disk arrays has increased the bandwidth of I/O data streams. This means the network could easily saturate when there are more than a few join operations. So, in our algorithm, bounded system resource consumption, especially utilization of main memory and balancing of I/O accesses and network transfers are considered. In our algorithm, holding the balance of system resource consumption takes precedence over memory utilization at pipeline processing level, so that the total performance is better than that of previous methods which give memory utilization preference.

The concept of a "balanced seed-tree(BST)" is introduced in order to balance system resource consumption all through the execution of the multi-way join in a shared-nothing environment. The balanced seed-trees are created along with generation strategies which first fully consume the network bandwidth and then attempt to utilize memory. The balanced seed-tree is the unit of parallel processing in which data streams flow through the network in a pipelined manner. The final scheduling tree is constructed from a combination of balanced seed-tree sets. Consequently, by using the balanced seed-tree, the balanced consumption of the bounded system resources, especially the bounded network bandwidth, for multi-way join processing on a shared-nothing architecture can be obtained.

2.1 Algorithm Overview

```
NEW ALGORITHM :
memsize = total amount of local memory of each node
relSET = all source relations
join_graphSET = all join graph edges
seedSET = pair of relations in join_graphSET
balanced seed-treeSET = resultSET = $\phi$
while(seedSET $\neq \phi$ ){
    get one_seed from seedSET;
    seedSET = seedSET - one_seed;
    balanced seed-treeSET = makeBST(one_seed);
    while (balanced seed-treeSET $\neq \phi$) {
        resultSET = makeresult(one_seed, balanced seed-treeSET);
        tmptree = costtree(resultSET);
        if( cost of tmptree < cost of minimum tree){
            minimum tree = tmptree
        }
    }
}

makeBST(one_seed) :
relset = all relations - one_seed;
while(relset != $\phi$) or (any relation of relset does not satisfies
the strategies listed below){
    find a combination of relation called "balanced
    seed-tree" from relset by using one_seed
    as a starting base
      strategy 1.
        generate a short right-deep tree in which the
        network bandwidth is fully consumed
        and whose hash tables and temporary result can
        be held in memory
      strategy 2.
        generate a short left-deep tree whose temporary
        result can be held in memory.
      strategy 3.
        generate a short right-deep tree in which all join
        operations can be processed in memory and
        whose temporary relation
        is written to disk and its cost is I/O dominant
      strategy 4.
        mark source relations which fit in memory
      balanced seed-tree_set +=
                              a generated balanced seed-tree
    relset = relset - a generated balanced seed-tree
}
return balanced seed-tree_set and the rest of relset

makeRESULT(balanced seed-treeSET) :
while(balanced seed-treeSET $\neq \phi$ ){
    make result tree in a bottom-up manner using
    a full search
}
return set of result trees
```

Table 1: Pseudo Code of the Proposed Algorithm

Table 1 describes the algorithm's processing flow. The proposed algorithm consists of a main loop and three modules:

- makeBST makes a balanced seed-tree set from the specified seed
- costtree calculates the cost of the specified trees and returns the minimum cost tree
- makeRESULT makes the result trees (or candidate trees) from a balanced seed-tree set using a full search.

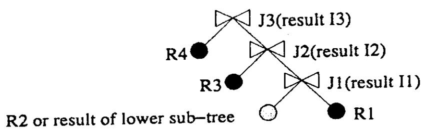

Figure 1: Balanced Seed-Tree

In the main loop, one pair of relations which are connected by an edge in the join graph is selected as a base for generating balanced seed-tree sets. Then, makeBST is called and the results (balanced seed-tree sets) of makeBST are passed as arguments of makeRESULT. The module of makeRESULT generates a final multi-way join plan from a balanced seed-tree set until all the given balanced seed-tree sets are processed. The minimum cost result is acquired by calling the costtree module which calculates the specified result trees generated by makeRESULT. The main loop is executed until all the pairs of relations joined by an edge are used.

The balanced utilization of system resources, especially balancing consumption of network transfers and I/O accesses is considered in order to generate a balanced seed-tree set, since, in a shared-nothing environment, parallel processing is not performed once the data stream is disturbed by excessive data transfer through the network or excessive I/O accesses. In [10], the dynamic programming method for parallel processing prunes the high cost branches in order to shrink the number of candidate trees, so almost all of the branches which store temporary result relations to disk will be pruned. In contrast, our method does not discard branches whose temporary result is written to disk as a candidate balanced seed-tree when the network bandwidth is relatively low or the network transfer cost is very large during the probe phase of a balanced seed-tree. In this case, although the cost of a balanced seed-tree seems large at this level, the total cost becomes optimal.

2.2 Balanced Seed-Tree

In this subsection, we describe how to generate a balanced seed-tree set from the given system resources and static database information.

In order to materialize the balanced usage of bounded system resources, we introduce four strategies for generating a balanced seed-tree by using two criteria : the consuming condition and the memory fitting condition. The consuming condition means that during the probe phase of each "balanced seed-tree", the cost of the data transfer through the network does not exceed the cost of reading the probe relation from disk and, if necessary, the cost of writing a temporary relation to disk. The memory fitting condition means that the left leaves of a "balanced seed-tree" fit into main memory or the temporary result will also fit into main memory. In order to evaluate these two criteria in the strategies, the balanced seed-tree cost formula are derived from system resources, especially overlapping conditions for I/O accesses and network transfers.

2.2.1 Cost of Balanced Seed-Tree

We consider the cost of a balanced seed-tree by using

Parameters	Description
N	number of processors and disks
$M(m)$	total memory size (memory size per node)
$R(r)$, $S(s)$	total size of relation R and S (sub relation size per node)
$\lvert x\rvert_{ioseq}$	cost of reading x tuples from the source relation
$\lvert x\rvert_{ioran}$	cost of reading x tuples from a temporary relation and writing the results
$\lVert x\rVert_{network}$	cost of transferring x tuples through the network between different nodes.
x_{move}	cost of moving x tuples from I/O buffer to network buffer after splitting tuples or from network buffer to hash table after hashing
x_{split}	cost of splitting x tuples
x_{hash}	cost of hashing x tuples
$y * x_{probe}$	cost of probing a hash table composed of y tuples with x tuples

Table 2: Notation for Cost Formula

Figure 1 as an example by using the notation presented in table 2. The shared-nothing system considered in this paper consist of N processors, $M = m \times N$ main memory and N disks.

The I/O accesses, the network transfers and the CPU processing can be almost overlapped in the hash-based join algorithm on a shared-nothing architecture. So, the cost of the build and probe phases is determined by the most dominant cost among the above three factors. Since the build and probe phases are executed independently, the cost of processing a balanced seed-tree is as follows:

$$\begin{aligned} Sub\ Tree\ Cost = {} & Build\ Phase\ Cost_{seed\_tree} \\ & + Probe\ Phase\ Cost_{seed\_tree} \end{aligned} \qquad (1)$$

First, if all the hash tables are generated from source relations R2, R3 and R4, the cost of the build phase is expressed as follows. Let $r_i (i = 2, 3, 4)$ be the size of the portion stored in each local disk of relation R2, R3 and R4 respectively. The total relation size $R_i (i = 2, 3, 4)$ is $r_i (i = 2, 3, 4) \times N$. We assume $R_2 + R_3 + R_4 < M$ and $r_2 + r_3 + r_4 < m$.

$$\begin{aligned} & Build\ Phase\ Cost_{seed\_tree} \\ = {} & Build\ Phase\ Cost_{J1} + Build\ Phase\ Cost_{J2} \\ & + Build\ Phase\ Cost_{J3} \\ \simeq {} & \max(|r2|_{ioseq}, ||r2||_{network}, \\ & r2_{split} + r2_{move} + r2_{hash} + r2_{move}) + \\ & \max(|r3|_{ioseq}, ||r3||_{nework}, \\ & r3_{split} + r3_{move} + r3_{hash} + r3_{move}) + \\ & \max(|r4|_{ioseq}, ||r4||_{nework}, \\ & r4_{split} + r4_{move} + r4_{hash} + r4_{move}) \end{aligned} \qquad (2)$$

If the temporary relation is generated by the lower balanced seed-trees on behalf of source relation R2, the cost of the build phase is expressed by the following two equations(3 or 4). When the temporary relation of the lower balanced seed-tree fits into main memory, the cost of generating a hash table is included in the cost of the lower balanced seed-tree. So, the cost of the build phase is as follows:

$$\begin{aligned} Build\ Phase\ Cost_{seed\_tree} = {} & Build\ Phase\ Cost_{J2} + \\ & Build\ Phase\ Cost_{J3} \end{aligned} \qquad (3)$$

When the temporary relation of the lower balanced seed-tree does not fit within main memory or the network transferring cost is large, the temporary relation is written to a local disk. Then, the temporary relation is read from disk and transferred through the network to the target node. In this case, the random I/O access cost is adopted to read the temporary results. Let *tempres* be the size of the intermediate results stored portion in each local disk. So, the cost of the build phase is as follows:

$$\begin{aligned} & Build\ Phase\ Cost_{seed\_tree} \\ = {} & Build\ Phase\ Cost_{J1} + Build\ Phase\ Cost_{J2} \\ & + Build\ Phase\ Cost_{J3} \\ \simeq {} & \max(|tmpres|_{ioran}, ||tmpres||_{network}, \\ & tmpres_{split} + tmpres_{move} + tmpres_{hash} + tmpres_{move}) \\ & + Build\ Phase\ Cost_{J2} + Build\ Phase\ Cost_{J3} \end{aligned} \qquad (4)$$

Although, in the build phase, I/O access and network transferring can be done in parallel for only one of the join operations, using the pipeline method, multi probe phases can be performed in parallel and it is necessary to consider the condition of overlapping between I/O access and network transfers of the probe relation(R1). When the temporary relation(I3) fits into main memory, the cost of the probe phase is as follows:

$$\begin{aligned} & Probe\ Phase\ Cost1_{seed\_tree} \\ \simeq {} & \max(|r1|_{ioseq}, \\ & ||r1||_{trans} + ||i1||_{trans} + ||i2||_{trans} + ||i3||_{trans}, \\ & r1_{split} + r1_{move} + r1_{hash} + r1 * r2_{probe} \\ & + i1_{split} + i1_{move} + i1_{hash} + i1 * r3_{probe} \\ & + i2_{split} + i2_{move} + i2_{hash} + i2 * r4_{probe} \\ & + i3_{split} + i3_{move} + i3_{hash} + i3_{move}) \end{aligned} \qquad (5)$$

Considering the balanced utilization of system resources in terms of total cost, it happens that it is efficient for the total cost to write back the temporary results to disk. The cost is as follows:

$$\begin{aligned} & Probe\ Phase\ Cost2_{seed\_tree} \\ \simeq {} & \max(|r1|_{ioseq} + |i3|_{ioran}, \\ & ||r1||_{trans} + ||i1||_{trans} + ||i2||_{trans}, \\ & r1_{split} + r1_{move} + r1_{hash} + r1 * r2_{probe} \\ & + i1_{split} + i1_{move} + i1_{hash} + |i1 * r3|_{probe} \\ & + i2_{split} + i2_{move} + i2_{hash} + i2 * r4_{probe}) \end{aligned} \qquad (6)$$

2.2.2 Strategies for Generating Balanced Seed-Trees

In this subsection, the strategies for generating a balanced seed-tree set are described in detail. There are four strategies which are applied for generating a balanced seed-tree by using the two criteria: the consuming condition and the memory fitting condition. In the first strategy (strategy 1), the strongest condition is applied to build a balanced seed-tree. That is, as for the consuming condition, the maximum network usage which does not exceed the I/O cost is employed and, as for the memory fitting condition, all the hash tables and the temporary results fit in memory. If there are relations which do not belong to the generated balanced seed-trees, looser restriction conditions are adopted in strategies 2, 3 and 4.

strategy1

This strategy is responsible for generating a seed-tree which fully consumes network bandwidth, even if a generated seed-tree does not utilize fully memory.

In order to generate a seed-tree whose shape is a kind of RD, find more than two relation combinations which have a common join edge and whose build phase hash tables and temporary results can be held in main memory. Then, using equation(5), find a base combination which satisfies the maximum network cost while the I/O cost is dominant for total cost. We also find other combinations which use less memory than the base combination, until the I/O cost is a dominant factor of total cost.

Until there are no relations which satisfy strategy 1, apply this strategy for generating balanced seed-trees.

strategy2

This strategy is responsible for boosting the potential memory utilization.

In principle, the shape of the balanced seed-tree is a kind of RD. However, when the joinability is small and the size of the intermediate relation becomes smaller than the source relation, a left-deep balanced seed-tree can be generated. Since it is not necessary for a short left-deep tree to hold multiple hash tables in memory, only a single hash table and a temporary result are held in memory. Although a short left-deep tree does not fully utilize memory, the chance of utilizing the rest of memory increases when a final scheduling plan is constructed from a balanced seed-tree set.

Find combinations of relations where a short left-deep tree can be composed from the remaining relations left over from strategy 1. Then find combinations which satisfy the condition that the maximum memory utilization of a generated short left-deep tree is smaller than the maximum size of any source relation which belongs to this left-deep tree. Until there are no relations which satisfy strategy 2, apply this strategy for generating balanced seed-trees.

strategy3

This strategy is responsible for transferring data streams through the network smoothly.

If there are some relations which do not satisfy strategies 1 and 2, find the combination of relations whose dominant cost is the I/O cost of reading the probe relation and writing the temporary relation to disk. The reasoning for this strategy is that once the network cost becomes a dominant factor, there are many tuples waiting to be transferred to the target processor and I/O accesses cannot be overlapped with network transfers.

This strategy is calculated from the two terms of equation(6), that is the I/O cost term (the first term) is larger than the network cost term. Until the I/O cost is dominant, find a combination of relations whose network cost is larger, but memory utilization is less.

Until there are no relations which satisfy strategy 3, apply this strategy for generating balanced seed-trees.

strategy4

This condition is responsible for providing information when makeRESULT makes the final scheduling tree.

If some relations do not satisfy all of the above strategies, find the combination of relations whose hash tables or result relation can be held in memory.

for the others

Any relations which do not satisfy the four condition are used in makeRESULT.

After adopting these four strategies, there are sets which consist of balanced seed trees generated by strategy 1, 2, 3 and 4, and the remaining relations. These sets are used to make a final plan.

3 Evaluation Results

In this section, we evaluate our method(BST) in comparison with previous work such as LD, RD, Segmented RD with MW and BC heuristics in [1] and deep-tree with BC heuristics (BCM) which utilize main memory for an intermediate relations proposed in [10]. Since the ZigZag tree proposed in [12] generates similar tree to BCM and it is not clear how to make up a ZigZag tree, we could no include it in this evaluation. As for the parallel dynamic programming method, we take a full search version of dynamic programming in order to get an optimal plan and to find the size of the search space in an exhaustive manner. If the condition of pruning a branch is changed, the quality and the search space of the dynamic programming method is also changed. However, the pruning condition is not clear in [10] and finding its condition is out of the scope of this paper. In order to evaluate the different parallel plans and the many different parameters, we generate synthetic databases and queries which are described in the following subsection.

3.1 Parameter and Simulation Environment

As the performance metric adopted for evaluating query plan is a normalized value which is defined as the ratio of the elapsed time of the generated plan to that of the optimal plan which is found through the exhaustive search. We calculate the elapsed time of each method using the proposed cost formula. The system parameters are shown in Table 3. We assume that the CPU power is 100 MIPS and it takes 1 clock ticks per 1 instruction. As for the parameters shown in Table 3, we take values from typical parallel machines and disks which are available today. The hash join algorithm parameters such as probe, split or hash are similar to those used in a previous study by Shekita, Young and Tan[10] and other hash-based join performance studies in [8, 6].

In this simulation, in order to extract the characteristics of each multi-way join plan, we generate the acyclic and cyclic join graph pattern. The join graph and join edge definitions are similar to those described in [1].

We generated 5500 sample databases as a test bed.

Description	Value
number of nodes	16
clock rate of individual processor	100MHz
total memory size	32MB
data transfer rate of sequential read from disk	4.8MB/sec
data transfer rate of random read and write to disk	2.4 MB/sec
data transfer rate through the network	24MB/sec
number of instructions to move a tuple from disk buffer (network buffer) to local memory	50
number of instructions to split a join attribute	100
number of instructions to hash a join attribute	100
number of instructions to probe an entry of hash table with a tuple	500

Table 3: Architecture and System Cost Table

The number of relations of each database is varied from 6 to 12 and the average number of relations is 8 which is the default parameter in [10]. The relation size varies from 10% of memory size to 200%. The join selectivity varies from 0.0001% to 0.01%. In this paper, the join selectivity means the ratio of the size of an intermediate relation to the product of the size of the two operand relations. In order to generate synthetic databases, the above three parameters are determined at random and the source relations are declustered over all disks uniformly. The join attribute of join operations are different from each other.

3.2 Effect of Bounded Network Bandwidth

In this subsection, we discuss how bounded network bandwidth affects the result trees generated by our algorithm. Figure 2 shows the scheduling trees of BST for varying network bandwidths. To clarify the following discussion, consider a very simple example query of 8 relations whose profile is shown in Figure 2. The sizes of all the both relations and intermediate relations are the same. Since the other algorithms such as LD, RD, MW, BC and BCM are not affected by network bandwidth, only the result of BCM is taken up as a comparative example. The result plan generated by BCM is also illustrated in Figure 2 to clarify the effect of network bandwidth. In Figure 2, the segments and the balanced seed trees(bst), which are the unit of parallel processing of BCM and BST respectively, are indicated by the dotted regions. The cost of each result trees is calculated by suming up the costs of each segment or bst.

Table 4 shows the total costs for the result trees and cost of I/O accesses and network transfers for each segment and bst. CPU cost is omitted in this table, since its cost is very small comparing with I/O and network costs. In order to easily compare the results of the two algorithms, both I/O access and network transfer costs are normalized by the I/O access time of one tuple. For example, the first column of segment 1 of BCM, which is the I/O cost of the build phase of segment 1, is 3000. That is, the I/O cost of the build phase of segment 1 is the same time required to read 3000 tuples from disk. Also, the second column of segment 1 of BCM, which is the network cost of the build phase of segment1, is

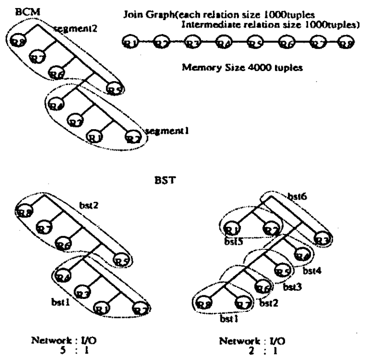

Figure 2: Generated Tree by Varying the Network Bandwidth

600. That is, the network cost of the build phase of segment 1 is the same time for reading 600 tuples from disk.

From Figure 2 and Table 4, we find that the shapes of the scheduling trees generated by our algorithm are changed by varying the network bandwidth. In this simulation, the shape of the result trees of the two algorithm are the same when network bandwidth is five times larger than the I/O bandwidth. However, when the network bandwidth becomes small, the result trees of our algorithm consists of several bsts. As shown in Table 4, since our algorithm intends to balance the consumption of bounded system resources, the I/O and network transfer costs of the probe phase of each bst is almost balanced and the I/O cost is dominant. On the other hand, the probe cost of segment 1 of BCM becomes the dominant network cost. Thus, when the network bandwidth becomes narrow, the cost of our algorithm is the same as the cost in the case of high network bandwidth. In contrast, the cost of BCM is deteriorated since BCM does not consider network transfer costs.

3.3 Quality of Result Trees

In this subsection, we report simulation results and show that our proposed method can generate higher quality scheduling plans in comparison to the other methods.

3.3.1 Quality of Result Trees under Bounded Network Bandwidth

The cost shown in Figure 3 shows the total simulation results. Since the cost of these results are normalized by the cost of the best plan, the cost expected for the best plan is 1. In Figure 3, each raw shows the results for each method and the box size depicts the percentage of queries whose normalized costs fall within the indicated range. The size of the black box in each row shows the percentage of generated plans which were optimal. The quality of generated plan of darker shaded boxes are better than lighter shaded ones. The size of the white one shows the percentage of plans whose cost is more than three times larger than the optimal plan. For example, we take the LD case. From the LD row, 59% of the cases generate the best

algorithm	segment# or bst#	Network Bandwidth : I/O Bandwidth							
		5 : 1				2 : 1			
		Build		Probe		Build		Probe	
		io cost	net cost	io cost	net cost	io cost	net cost	io cost	net cost
BCM	segment 1	3000	600	1000	800	3000	1500	1000	2000
	segment 2	3000	600	2000	800	3000	1500	2000	2000
	total				9000				10000
BST	bst1	3000	600	1000	800	1000	500	1000	1000
	bst2	3000	600	2000	800	0	0	1000	1000
	bst3	–	–	–	–	0	0	1000	1000
	bst4	–	–	–	–	0	0	1000	1000
	bst5	–	–	–	–	1000	500	1000	1000
	bst6	–	–	–	–	0	0	2000	1000
	total				9000				9000

Table 4: Cost of Result Trees by Varying the Network Bandwidth

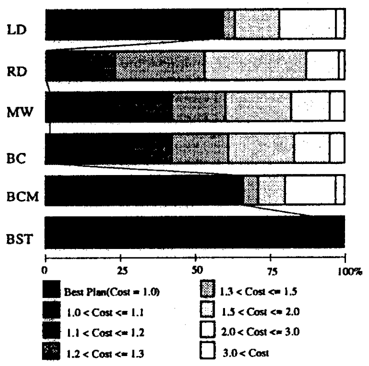

Figure 3: Quality of Generated Plans
- the case of bounded network bandwidth -

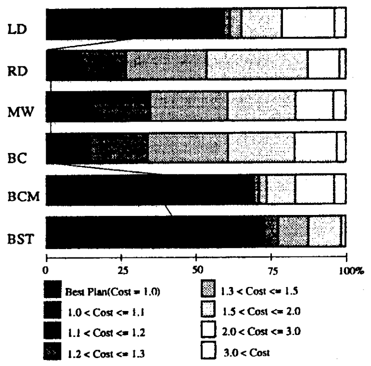

Figure 4: Quality of Generated Plans
- the case of the unbounded network bandwidth -

plan and 3% of the cases generate a plan whose cost is 3 times larger than that of the best case.

From this figure, we find that 90% of the results generated by our proposed algorithm are the best plan. Moreover the cost of the worst plan generated by our method is 1.3 times smaller than that of the best case, although the cost of the worst plan generated by the other methods is 3 times larger than that of the best case. From this observation, the quality of generated plans of our method is superior to the others. It is important for an optimizer to not only generate the best plan regularly but also not to make bad plans. From Figure 3, our algorithm can be assured to have the strength to avoid generating bad plan.

On the other hand, more than half of the results generated by RD, MW and BC are not good plans, since their cost is 1.5 times larger than the optimal one. This is because these methods do not consider holding an intermediate relation in main memory for the hash table of the next sub-tree.

One point of interest which can be observed from this table is that results generated by LD and BCM are either the best plan or a bad plan and there are few intermediate plans generated by either algorithm. This is because LD often assigns a relation whose size is small to the bottom of the tree so that the size of source and intermediate relations becomes larger near the top of the tree and the potential of their hash table overflowing memory becomes high. BCM's result are similar to those of LD, since when the size of temporary relations becomes large and the temporary relations do not fit into memory, the generated tree of BCM resembles the result of LD in tree shape.

3.3.2 Quality of Result Trees with Unbounded Network Bandwidth

In this subsection, we show the simulation results for unbounded network bandwidth. The unbounded network means that the network bandwidth is sufficient to handle all of the data streams smoothly. In this case, our method's balanced resource consumption is not expected to provide significant improvements.

The cost shown in Figure 4 shows the simulation results. In this figure, the cost of the results are also normalized as in the above figures. In Figure 4, each row shows the results for each method and the shaded areas depicts the percentage of queries whose normalized costs fall within the indicated range. The size of the black region in each row is the percentage of the total generated plans which were the best plan.

From Figure 4, we find that the percentage of generated best plans by our proposed algorithm decreases in comparison with Figure 3. The percentage of the

TYPE	EXHAUSTIVE	OUR METHOD(BST)	%
total	80640	235	0.29
type 1	89230	276	0.34
type 2	46080	195	0.24
type 3	92160	374	0.41
type 4	161280	377	0.23

Table 5: Search Space

best plan of our algorithm is almost the same as that of BCM. However, the cost of the worst plan generated by our method is 2.0 times larger than that of the best case, although the cost of the worst plan generated by the other methods is 3 times larger than that of the best case. Thus, our algorithm is efficient when the network bandwidth is unlimited. This is because we consider not only network bandwidth but also the utilization of main memory in order to introduce the restriction conditions for generating balanced seed-trees. Thus, our algorithm can generate slightly better plans than previous work when it cannot take advantage of balancing the network transfers and I/O accesses.

3.4 Search Space

The size of the search space is another important factor for the optimization of multi-way joins. Of course, the quality of generated tree is improved when the search space becomes large. However, as pointed out by [5], the explosion of the search space for the optimization results in the algorithm being unrealistic. Although our main objective for the proposed algorithm is to manage system resource consumption by considering network bandwidth, we show that the search space of our algorithm is small enough to be practical in this subsection.

Table 5 shows the search spaces of our method and an exhaustive method. The values in this table is the average of the simulation results. From Table 5, we find that the search space of our method is very small in comparison with an exhaustive method. Comparing the results of our method for type 1 and type 2 databases, the search space of type 2 is smaller than that of type 1, since the relation size of type 2 is large and the number of balanced seed-trees decreases from using the restriction condition of memory utilization. As for the results of type 3 and type 4, since the strategies for reducing a join graph are more numerous than those of type 1, the search space increases. Especially, the search space of an exhaustive method increases drastically when the number of join edge increases. However, the search space of our method for type 4 does not increases rapidly, since the restriction conditions suppress generating too many candidate seed-trees.

4 Conclusions

In this paper, we consider multi-way join processing in a shared-nothing environment and propose how multi-way join scheduling is constructed efficiently from a given resource environment such as memory size, network transfer and CPU processing costs. From the point of view of large database systems on a shared-nothing system these days, it is important for pipeline processing of multi-way joins to transfer large amount of data through the network smoothly. Our algorithm first generates balanced seed tree seeds which fully consumes the network bandwidth for pipeline processing. Then, these balanced seed tree sets are combined with each other to finally produce an optimal query tree.

The proposed new multi-way join plan is evaluated in comparison with previous work by using the introduced cost formula. From the evaluation results, not only is the quality of the proposed method better than previously presented algorithms such as left-deep, right-deep and segmented right-deep trees, but the quality of our algorithm does not deteriorate comparatively. Especially, when the network bandwidth is bounded, our algorithm is more robust than other methods which do not take into account the network bandwidth. Moreover, even if the network bandwidth is not limited, our algorithm generates more best plans than the other methods and the quality of the proposed algorithm does not deteriorate. Although our method can generate numerous best plans, the search space of our algorithm is small enough to be practical in comparison with the exhaustive method.

We are planning to implement our algorithm on a commercial parallel machine, which is Fujitsu AP-1000, and investigate the effect of overlapping the I/O accesses and the network transfers on a real machine.

参考文献

[1] Chen, M., Lo, M., Yu, P.S., and Young, H.C.: Using Segmented Right-Deep Trees for the Execution of Pipelined Hash Joins Proc. of VLDB, pp.15-26 (1992)

[2] Dewitt,D.J., and Gray,J. : Parallel Database Systems: The Future of High Performance Database Systems, ACM Comm., Vol.35, No.6, pp.85-98(1992)

[3] Dewitt,D.J., Katz, R., Olken, F. and et al.: Implementation Techniques for Main Memory Database Systems, Proc. of SIGMOD, pp.1-8(1984)

[4] Hong, W. and Stonebraker, M.: Optimization of Parallel Query Execution Plan in XPRS, Journal of Parallel and Distributed Database Systems, pp.9-32(1993)

[5] Lanzelotte, R.S.G., Valduriez, P., Zait, M.: On the Effectiveness of Optimization Search Strategies for Parallel Execution Spaces, Proc. of VLDB, pp.493-504(1993)

[6] Lu, H, Tan, K.and Shan,M: Hash-based Join Algorithms for multiprocessor computer with shared memory, Proc. of VLDB, pp.549-560(1990)

[7] Lu, H, Tan, K.: Optimization of Multi-Way Join Queries for Parallel Execution, Proc. of VLDB, pp.549-560(1991)

[8] Schneider, D.A. and DeWitt, D.J.: Tradeoffs in Processing Complex Join Queries via Hashing in Multiprocessor Database Machines, Proc. of VLDB, pp.469-480(1990)

[9] Selinger, G., et al: Access Path Selection in a Relational Database Management System, Proc of SIGMOD, pp.189-222(1979)

[10] Shekita, E.J., Young,H.C. and Tan, K.L.: Multi-Join Optimization for Symmetric Multiprocessor Proc. of VLDB, pp.479-492 (1993)

[11] Srivastava, J. and Elsesser, G.: Optimizing Multi-Join Queries in Parallel Relational Databases, Proc. of DE, pp.84-92 (1993)

[12] Ziane, M., Zait, M., and Borla-Salamet, P.: Parallel Query Processing with Zigzag Trees, Journal of VLDB,Vol.2, No.3, pp.277-302(1993)

[13] Ziane, M., Zait, M. and Boral-Salamet, P.: Parallel Query Processing in DBS3, Proc. of DE, pp.93-102(1993)