Efficient and Accurate Cost Models for Parallel Query Optimization

(Extended Abstract)

Sumit Ganguly *
Rutgers University

Akshay Goel
Rutgers University

Avi Silberschatz
AT&T Bell Labs

Abstract

One of the key problems in the design of a query optimizer for parallel databases is the derivation of efficient and accurate cost models. These cost models rely on accurate estimators for the response time of a query, which can only be obtained by invoking expensive scheduling algorithms. To achieve efficiency, cost models must estimate the response time using approximation functions. In this paper, we consider the issue of design and validation of cost models for SQL-query optimizers for parallel executions. We present a theoretical foundation underlying the design of efficient cost models that accurately approximate response time. Based on the above theoretical foundation, we formulate two heuristical cost functions. We use simulations to determine the accuracy of these heuristic functions. Our simulation results indicate that the heuristic cost functions generate plans whose performance is within 20-60% of the optimal scheduled plan for 90-95% of the queries.

1 Introduction

The primary goal of a query optimizer in a parallel relational database system is to come up with an execution plan, for a given query, which will reduce the response time of the query. The response time of a given execution plan is the amount of time it takes to complete the execution of the plan on the given parallel machine.

For a given query, there are a large number of execution plans that are equivalent to each other; that is, they produce the same answer for the query, as dictated by the semantics of the SQL language. There exists one execution plan, among the set of all execution plans, for a given query, which has the least response time. The goal of query optimization is to find this optimal execution plan. Following the time honored path of database query optimization, one would use a *cost model* (also called *cost functions*) to estimate the response time of a given query plan and then search the space of query plans in some fashion to return a plan with a low (hopefully minimum) value of cost. The accuracy of the search procedure as well as its complexity would therefore be strongly influenced by the accuracy of the cost model and the computational complexity of the cost model.

The query optimization problem for a parallel database system, where queries are executed serially one at a time (but each query is executed in parallel), may be viewed as follows. The query optimizer is given as an input a single SQL query, the specification of the parallel architecture on which the query is to be executed on, and the layout structure of the database data. The output is the optimal execution plan.

One way of tackling this problem is to take the set of *sequential execution plans* for the given query – plans that are appropriate for the query in a sequential database system where no parallel execution is allowed, and converting this set into a set of *parallel execution plans* – plans that are appropriate for the query in a parallel database system. This is done, by specifying which resources should be allocated to each operation in the sequential plan and the starting times of each operation. In this way, one can view a parallel execution plan as encoding both a sequential execution plan and a schedule for the execution of the plan on the given parallel machine. In other words, a parallel execution plan = a sequential execution plan + initial data layout + scheduling information. This makes the query optimization problem for parallel databases more complicated than the corresponding query optimization problem in sequential databases.

Given the above, one straightforward way of computing the optimal parallel execution plan is the following. Suppose that one can devise an algorithm A, that takes a sequential execution plan and a parallel machine description and returns the time taken by the least response time schedule for that plan among the all pos-

*This work is partially supported by an NSF grant No. 4-21096

PODS '96, Montreal Quebec Canada
© 1996 ACM 0-89791-781-2/96/06..$3.50

sible parallel executions. Using algorithm A, one can exhaustively search the space of all *sequential plans* to find the best *parallel execution* plan.

Although this approach will derive an optimal execution plan, it nevertheless, is not a viable approach in its current form since, there is no known algorithm A that runs in polynomial time. This efficiency problem has so far been addressed by using a polynomial-time algorithm B, which is intended to approximate the performance attained by A. The various approaches for deriving B, which have been used in the literature, can be classified as falling into one of the two classes.

1. B is an ad-hoc function whose accuracy is not argued either by theoretical arguments or by experimental validation.

2. B estimates the minimum response time of a sequential plan on a parallel machine by using a resource allocator. The resource allocator takes a sequential plan (together with estimates on the sequential running times of each operators, data-dependency information, size of intermediate values, etc.) and produces a schedule for the parallel execution of the given execution plan. An estimate for the completion time of this schedule is computed and is used as the approximation to the output of algorithm A.

Both approaches suffer from some major deficiencies. The problem with the first approach is that the accuracy of the estimator functions is not objectively justified. The second approach is sound, although, computationally expensive. Algorithms for resource allocation to sequential plans are known to have a complexity of $O(n \cdot m)$ or higher, where n is the number of resources available in the parallel machine and m is the size of the sequential execution plan. If we were to optimize for a sequential database system, then the time required to estimate the performance would be $O(m)$. For a parallel database system, however, n could be in the hundreds and possibly in the thousands. Thus, the time taken to optimize a query for parallel executions can potentially be several orders of magnitude worse than the time it takes to optimize for sequential executions.

We propose an alternative approach for estimating the minimum response time of parallel execution of a given sequential execution plan. Our approach is to estimate the response time by using inexpensive estimator functions that *do not compute a schedule for allocating resources to the sequential plan.* Our estimator functions can be computed much more efficiently than actual resource scheduling algorithms. As a result, we are able to:

1. Derive parallel query optimization techniques that have a complexity which is comparable to sequential query optimization and

2. Generate parallel execution plans that are close to the optimal plans (in terms of response time) with a high probability.

We present theoretical arguments that show that our proposed estimator functions estimate response time quite accurately and efficiently. We show, by experimentation, that our cost functions yield query plans, which for 95% of the queries, are within 20-60% of the optimal performance when tested against accurate scheduling function. This implies that if our cost functions are used to estimate the response time of a sequential execution plan on a parallel machine, then for 95% of the queries, the best plan obtained shall have a response time that is no worse than 1.2-1.6 times the response time of the best plan obtained using accurate scheduling functions.

Our work was done in the context of *shared nothing* architecture. We note, however, that the cost models which we are advocating in this paper are appropriate for optimizing queries for shared memory parallel architectures as well. This issue, however is beyond the scope of this paper.

The remainder of the paper is organized as follows. Section 2 presents a review of the cost models. Section 3 presents the communication cost model used in the paper. In Section 4, we present estimator functions for response time optimization. Section 5 presents the cost models used for experimentation. Section 6 discusses the simulation and its results. Concluding remarks are offered in Section 7.

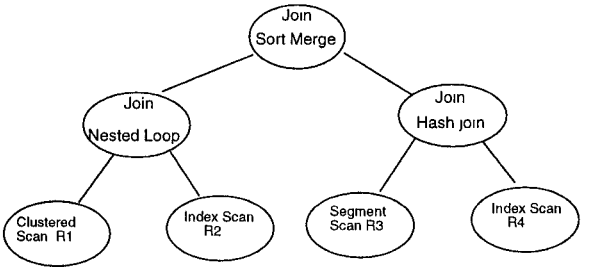

Figure 1: Example of an Execution tree

2 Review of Cost Models

To define cost models, we use the notion of an *operator tree* [Gan92, GHK92, Hon91, SriEls93], which is is an extension of an execution tree [SAC+]. An execution tree is a tree of nodes, where each node is annotated to represent a specific method of joining the operands and of accessing the relations (See Figure 1). Each join operation may be a composition of several operations;

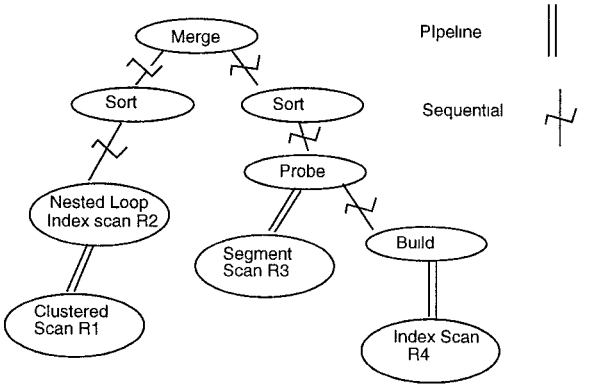

Figure 2: Example of an Operator tree

for example, sorting and merging files, building a hash table and probing it. An operator tree is an expansion of each node of the execution tree into its constituent operations (See Figure 2). Data dependency between adjacent operations in the operator tree is placed on the edges of the tree. We consider two kinds of data dependencies between operations: *sequential* and *pipelined*. Sequential dependency arises in situations where a consumer must wait until the producer finishes its execution. For example, when a producer or a consumer is a "sort" operation, or when the producer is a "build-hash-table" operation and the consumer is a "probe-hash-table" operation. Pipeline dependency arises in situations where the consumer can start consuming before the producer finishes its execution. For example, when the producer is a "merge" operation and the consumer is a "probe" operation, or when the producer is a "probe-hash-table" operation and the consumer is the "index-scan" of a "nested-loop" operation. Since the implementation properties of relational operators are well known, it is easy to construct a table of data dependencies between all pairs of relational operators [Gan92, GHK92].

Cost models for parallel query optimization are influenced by the number of possible ways in which resources may be allocated to the execution plans. Existing cost models either actually schedule operators onto resources and then calculate the response time [LVZ93, LST91, GHK92], or assume that all resources are used to execute each operator [Hon91, SYT93]. The issue of designing efficiently computable and accurate estimator functions for response time has not been addressed earlier.

3 Communication Cost Model

In this section, we briefly describe the communication cost model used in the subsequent sections of the paper. We assume that there is a centralized cost model (similar to the System R cost model) that estimates:

- The time of computation of each of the operators in an operator tree on a sequential machine.
- The sizes of the inputs and the outputs of each of the operators.
- The memory requirements of each of the operators.

We assume a shared-nothing architecture consisting of processors $P = \{P_1, P_2, \ldots, P_n\}$, which communicate with each other via a switched network. Each P_i is connected to a local storage unit (e.g., a single disk, multiple disks, or a RAID disk). We assume that the processors are identical to each other and that the disks are identical to each other. Operators may be parallelizable; that is, they can be executed in parallel using multiple processors. Only relational-scan operators cannot be arbitrarily parallelized, since they must fetch data from the disks that store them.

3.1 Cost of Data Transfer

Consider an operator tree where the output of operator o_i is consumed by operator o_j. Let us assume that o_i and o_j are computed using the set of processors Q_i and Q_j respectively. Let $send_{ij}$ denote the cost incurred by each processor in Q_i (assumed by symmetry, to be equal) in sending its output to the processors in Q_j. Similarly, let $receive_{ji}$ denote the cost incurred by each processor in Q_j (assumed again to be equal) in receiving the data sent from the processors in Q_i. The term $delay_{ij}$ denotes the time spent in the network by the message sent by any processor in Q_i before it reaches its destination processor in Q_j. Let Z_i denote the set of processors across which the relations involved in the operation o_i have been partitioned and let Z_j denote the set of destination processors across which the relations are redistributed. We assume that the partitioning of the relations in both Z_i and Z_j is done in a uniform manner.

Data partitioned parallelism of relational operators is effected by choosing a partitioning attribute(s). For example, consider the join of two relations $R(a, b, c)$ and $S(d, e, f)$ with the join predicate $R.b = S.d$. In this case, it is natural to use b as the partitioning attribute of R and d as the partitioning attribute of S. In this case, a hash function h is chosen and applied to the b or d attributes of each tuple in R and S respectively. If the return value of the hash function h is k, then the tuple is sent to the k^{th} processor. This ensures that tuples from R and S that are sent to distinct processors have no possibility of contributing the join.

Suppose now that the output of the above join of R and S (i.e., $R \bowtie S$) is joined with another relation $T(j, k, l)$ with the join predicate $S.f = T.j$. In this case, the partitioning attribute for this join would be f for $R \bowtie S$ and j for T. Now, $R \bowtie S$ is already

partitioned by b (or d, since $b = d$), and needs to be repartitioned on a different attribute, namely f. In contrast, suppose that the join predicate is $S.d = T.j$. In this case, the output of S is already partitioned on d. The latter case is called the *one-to-few* case and the former case is called the *all-to-all* case. To justify this naming, suppose that $R \bowtie S$ is scheduled on processors $\{P_1, P_2, \ldots, P_k\}$ and the second join is scheduled on processors $\{P_{k+1}, P_{k+2}, \ldots, P_{2k}\}$. Then, for $1 \leq i \leq k$, processor P_i has all tuples of $R \bowtie S$ in which $h(b) = h(d) = i$. If we choose the hash function for the final join as $h'(x) = h(x)+k$, then we see that, in the one-to-few case, tuples in P_i are sent only to P_{k+i}. In the all-to-all case, tuples in P_i have to be rehashed on the new attribute j and sent to, possibly, each processor in P_{i+k}, for $1 \leq i \leq k$[1].

The cost of communicating a single message of B bytes from a sender to a remote receiver is modeled as $t_{send} = t_{receive} = \alpha + B\beta$ where α is the (constant) operating system call overhead for message startup; that is, the processor-network interface time before any data is placed onto network, and β is the per-byte cost of transmitting the message (message volume component) into network channel. The values of α and β for typical commercial machines range between 1000-3000 *CPU* cycles and 1 cycle/byte-1 cycle/word respectively. We derive cost formulae for communication under two kinds of buffering mechanisms; namely, exclusive buffering and shared buffering.

3.2 Exclusive and Shared Buffering

There are two kinds of buffering mechanisms for communication – exclusive buffering and shared buffering. In this section we describe the communication model for each of these buffering schemes.

Let Q_i be the set of processors used for executing operation o_i. Let Q_j be the set of processors executing the successor operation o_j. That is, o_i is the producer and o_j is the consumer operation. In the exclusive buffering scheme it is assumed that each of the processors in Q_i executing o_i has $|Q_j|$ distinct buffers (one for each processor in Q_j), each buffer of size B. Whenever a processor in Q_i wishes to send a tuple to some processor P_k in Q_j, it adds the tuple to the buffer corresponding to processor P_k. Once a buffer becomes full, it is sent to its destination. The expressions for $send_{ij}, receive_{ji}$ and $delay_{ij}$ are presented in the Appendix.

In the shared buffering scheme, each processor in the parallel machine consisting of n processors has $n-1$ buffers of size B, one for each of the remaining $n-1$ processors in the parallel system. Whenever a query wishes to send a message from processor P_i to processor P_j $(j \neq i)$, it adds the message to the j^{th} buffer at processor P_i. Buffers are sent when they are full. More details of this scheme can be found in [GGMW]. It is shown in [GGMW] that the cost of transferring a message of K bytes is approximated by $\lceil K/B \rceil \cdot B \cdot \gamma$ where γ is 2β and $B \geq \alpha/\beta$.

[1]It is possible that the functional dependency $F \leftrightarrow D$ is satisfied in S. In that case, the all-to-all case reduces to the one-to-few case)

4 Response Time Estimators

In this section, we present a solution to the basic problem considered in the paper; namely, the design of efficient and accurate estimator functions of response time. We design two estimator functions G and H, which are provable accurate and efficiently computable.

We approach the problem in two steps. First, we assume that the *approximate degree of parallelism* of each operator is known; that is, the number of processors x_i that are used to execute node i for each i in the tree is known. Using this information, we efficiently and accurately estimate the optimal (or scheduled) response time of the tree. Secondly, we approximate the *optimal degree of parallelism* of each node in the operator tree.

To define the estimator functions G and H, we must first define two functions A and C. For a given operator tree T, $A(T)$ measures the average processing work done by a processor. The function $C(T)$ (critical-path length) measures the length of the longest path from the root to a leaf in the tree.

The following notation is used to define $A(T)$ and $C(T)$:

- x_i — Degree of parallelism of node i in the operator tree
- t_i — Sequential processing time of node i on each of the x_i processors
- $PRED_i$ — The set of predecessor nodes of i sending data to node i; that is, there is an edge from each node in $PRED_i$ to node i.
- $SUCC_i$ — The set of successor nodes of i receiving data from node i
- $delay_{ui}$ — Denotes the time spent by the messages in network from the processors executing the nodes $u \in PRED_i$ to the processors executing node i
- n — Number of processors in the parallel machine

- m Number of nodes in the operator tree
- δ Network delay cost per byte

We are now in a position to formally define $A(T)$ and $C(T)$.

1. Let T_i denote the total cost incurred by node i in the operator tree, which is the cost of receiving data from the processors executing the predecessor nodes, plus the processing cost at node i, plus the cost of sending the data to the processors executing the successor nodes after redistribution. Formally,

$$T_i = \sum_{u \in PRED_i} receive_{iu} + t_i + \sum_{v \in SUCC_i} send_{iv}.$$

2. Let y_i denote the completion time of node i in the operator tree, which is the maximum completion time among the predecessors of node i, plus the delay time for the messages from processors executing the predecessor nodes to reach the processors executing node i, plus the *total cost* incurred by node i. Formally,

$$y_i = max_{u \in PRED_i}(y_u + delay_{ui}) + T_i.$$

3. $A(T)$ denotes the average work per processor, which is the total work done by all the nodes divided by the number of processors in the machine. Formally,

$$A(T) = (1/n) \cdot \sum_{i=1}^{m} T_i \cdot x_i$$

4. $C(T)$ denotes the critical path time, which is the max completion time of the nodes in the operator tree. Formally,

$$C(T) = max_{i=1}^{m} y_i.$$

Given $A(T)$ and $C(T)$, we are finally in a position to formally define the estimator functions H and G, as follows

- $G(T) = max(A(T), C(T))$.
- $H(T) = A(T) + C(T)$.

We now state several basic properties.

Theorem 1 : *Let T be an operator tree to be scheduled on a given set of n processors. Let b be an upper bound on the degree of parallelism of any node of the operator tree. Let $R_{opt}(T)$ be the response time of T using the best resource scheduling algorithm. Then,*

$$\begin{aligned} G(T) &\leq R_{opt}(T) \leq \left(1 + \tfrac{n}{n-b+1}\right) \cdot G(T) \\ 0.5 \times H(T) &\leq R_{opt}(T) \leq \left(1 + \tfrac{n}{n-b+1}\right) \cdot H(T) \end{aligned}$$ □

The proof of the Theorem 1 is derived using Theorem 1 in [RSB94]. Note that G and H functions are computable in time $O(m)$. The problem of computing the optimal schedule time is NP-complete and the best-known approximations involve the LSA heuristic [GGJ78, Gra69, RSB94, TWY92, WC92] (or its variants) with a cost of approximately $O(n \cdot m)$. We note again that G and H are provably good estimators of response time (especially when $b \leq n/2$) and can be computed very efficiently.

We now address the problem of efficiently computing a good approximation to the degree of parallelism of each node i. The degree of parallelism using the shared buffer model is denoted by $x_s(i)$ and using the exclusive buffer model is denoted by $x_e(i)$. Let B_i denote the size of the input + output of node i (i.e., $B_i = \sum_{j \in PRED_i} B_{ji} + \sum_{k \in SUCC_i} B_{ik}$). We say that an operator is *coarse-grain* with respect to the exclusive buffer model if $B_i \cdot \beta \leq t_i$. An edge $i \rightarrow j$ is said to be *coarse-grain* with respect to the shared buffer model if $B_{ij} \cdot \gamma \leq (t_i \cdot t_j)^{1/2}$. An operator tree is said to be coarse-grain with respect to the exclusive buffer model and shared buffer model if each node (respectively, each edge) is coarse-grain. Coarse-grain operator trees may be expected to be widely found among practical database execution trees.

Let L_i denote the number of neighboring nodes of node i in the operator tree with whom the communication of node i uses the one-to-few communication pattern. Let N_i denote the number of neighboring nodes of node i with whom node i uses the all-to-all communication pattern. And let B denote the size of the buffer pages of the processors. We now present the formulae for computing the degree of parallelism of a node i and then argue the goodness of the approximation formula in the following theorems.

$$x_e(i) = \begin{cases} min((t_i + B_i \cdot \beta)/L_i, n) \\ \quad \text{if } N_i \cdot (t_i + B_i \cdot \beta) < L_i^2 \\ min((t_i + B_i \cdot \beta)/N_i)^{1/2}, n) \\ \quad \text{otherwise} \end{cases}$$

$$x_s(i) = \begin{cases} min(\lceil (t_i/(B \cdot \gamma)^{1/2} \rceil, n) \\ \quad \text{if } N_i \geq 1 \\ min(\lceil (t_i/(B \cdot \gamma) \rceil, n) \\ \quad \text{otherwise} \end{cases}$$

Since network propagation is assumed to be concurrent with processing, it is not unreasonable to assume that at low network congestions, the network delay time $B \cdot \delta$ is not significant compared to the processing time. We therefore ignore the delay time in the following discussions.

Given an operator tree to be scheduled over n processors. Let A_s^{opt} and C_s^{opt} denote the minimum average work per processor and the minimum length of

the critical path possible, respectively. Let A_s^h and C_s^h denote the average work and critical path length using the definition for x_s above.

Theorem 2 : *Given an operator tree that is coarse-grain with respect to the shared buffer model,* $A_s^h \leq 4 \cdot A_s^{opt}$ *and* $C_s^h \leq 2 \cdot C_s^{opt}$. □

To discuss the goodness of the approximation scheme for x_e we must define the K-function as follows. For a given operator tree T, $K(T) = \sum_{i=1}^{m} t_i$. Let K_e^{opt} denote the minimum possible value of K obtained by varying the degree of parallelism of each operator. Let A_e^{opt} denote the minimum value of average work and let K_e^h and A_e^h denote the values of $K(T)$ and $A(T)$ using the formulae for x_e.

Theorem 3 : *Given an operator tree that is coarse-grain with respect to the exclusive buffer model,* $K_e^h \leq 2 \cdot K_e^{opt}$ *and* $A_e^h \leq 4 \cdot A_e^{opt}$. □

The above theorems lend credence to the assertion that the G and H estimator functions are accurate estimators of the optimal response time of a given operator tree. It is easy to see that the complexity of both G and H functions is $O(m)$, whereas the complexity of well-known heuristics such as LSA is $O(n \cdot m)$. Thus, the complexity of G and H is significantly less than the complexity of LSA.

Consider the set $\mathcal{T} = \{T_1, T_2, \ldots, T_r\}$ consisting of all the operator trees associated with some query to be optimized. Let O_L be an algorithm that finds the tree T_i in $\mathcal{T}$ with the least response time. Let O_G (O_H) be an algorithm that finds the tree T_j in $\mathcal{T}$ with the least response time with respect to G (H). If we assume that the search engine for O_L, O_G and O_H are the same, then we can infer the following.

1. Algorithm O_G (O_H) is significantly more efficient than algorithm O_L.
2. The optimal tree returned by O_G (O_H) has a performance that is close to the performance of the optimal tree returned by O_L, by virtue of the theoretical arguments stated above.

We conclude from the observations and theorems above that the functions G and H can be expected to be accurate and efficient predictors of response time in very general settings. A general principle behind the design of accurate estimators of response time is to incorporate a combination of the critical-path and the average work per processor.

5 Experimental Cost Functions

In this section, we discuss the heuristic cost functions that were used as efficient estimators for response time and a more refined cost function that scheduled a given operator tree to obtain an estimate.

5.1 Heuristic Cost Functions

In order to estimate the accuracy of G and H, in a realistic setting, we generalized the above model to estimate CPU and disk costs separately (effectively, this creates pairs of cost functions, G_{CPU} and G_{disk}, H_{CPU} and H_{disk}) and to include cost formulae to estimate cost of pipelined computation. The model for disk usage can estimate costs of non-uniform accesses to database relations, which occur naturally when base relations are range-partitioned. The degree of parallelism of each operator is computed except for scan operators. The extensions are straightforward and are similar in nature to the computations presented in [LVZ93]. However, the novel feature of our cost model is the incorporation of the critical-path into the cost model that makes the estimate efficient. Due to lack of space, we do not present the full details of the the cost model here. These are presented in [Goel95].

5.2 Scheduling Estimator Function

In order to experimentally measure the accuracy of our proposed estimator functions, we compared our estimates for response time to the SCH function, which makes a more refined estimation of response time of a given database execution tree by scheduling the tree. The scheduling algorithm used is a variation of the scheduling algorithm proposed in [NSHL93, TL94]. Let S be an operator tree (actually, S is assumed to be a segmented tree, obtained by coalescing neighboring pipelined nodes into single nodes, subject to memory availability). The schedule is computed as follows.

1. Create k shelves, where k is the length of the longest chain in S. The length of a chain is the number of operators in the chain. Place the i^{th} operator of a longest chain on the i^{th} shelf. We assume that a parent node is placed on a higher shelf than its child [NSHL93].
2. Place other operators on the shelfs according to some heuristic, such as lowest-level-first [Goel95] heuristic or the balanced work-per-shelf [NSHL93] heuristic.
3. The degree of parallelism is calculated by balancing the workload (i.e., give one more processor to the most time-consuming operator, continue until all processors are occupied) [TWPY92, BB90].

The completion time of a schedule is estimated as the sum of the completion times of each of the shelves in the schedule. Since within a shelf, the assignment of processors to operators is completely specified, the time taken to complete the work on each processor is estimated. The completion time of a shelf is estimated as the maximum of the completion times among all the processors.

6 Simulations

In this section, we measure, using simulation, the inaccuracies that result from using the G and H functions rather than using the more detailed cost function SCH. We first describe an overview of the simulations conducted and the measured quantities. We then present the results of the simulations and our conclusions.

6.1 Overview

The simulations were designed to see the impact of scheduling on the accuracy of the cost models; all other factors (such as estimation of selectivity, intermediate relation sizes, etc.) remaining the same. However the set of queries over which the simulations were done, had variation in these parameters to simulate all types of queries. The primary goal of the simulations was to see how well each of the optimal plans with respect to the G and H cost functions performed as compared to the more detailed cost model SCH. A secondary goal of the simulations was to see how well the traditional *work* cost function (used to estimate sequential query time) performs with respect to SCH.

The simulations consisted of taking conjunctive SQL queries (SPJ queries) and enumerating the set of all plans for that query. For each query, the optimal plan with respect to each of the cost function was found. Thus, for a query Q, we obtain the plans, $G_{opt}(Q), H_{opt}(Q), SCH_{opt}(Q)$ and $work_{opt}(Q)$. Then, we run the SCH function, which is the detailed scheduling function to obtain an estimate of the response time of each of the plans generated. For example, for a query Q, if we run the SCH algorithm on the plan $G_{opt}(Q)$ we obtain the response time of the plan $G_{opt}(Q)$. We then compute the ratio

$$\frac{SCH(G_{opt}(Q)}{SCH(SCH_{opt}(Q))}$$

This ratio is always constrained to be ≥ 1, since the plan $SCH_{opt}(Q)$ is optimum with respect to the SCH function over the space of all plans for Q. However, the ratio above measures the extent of error incurred by finding the optimal G plan instead of the optimal SCH plan for the given query Q. Similarly, we compute the ratios $SCH(H_{opt}(Q)/SCH(SCH_{opt}(Q))$ and $SCH(work_{opt}(Q))/SCH(SCH_{opt}(Q))$ to estimate the errors involved in finding the optimal plan using the respective functions rather than the SCH function.

6.2 Simulation Results

The simulations were performed in sets of queries over 4 relations, 5 relations, 6 relations and 8 relations.

The complete coverage of our cost model, simulation setup, architectural parameters, the software parameters used in estimating sequential execution time, and the graphs of the simulation have been discussed in [Goel95]. The details have been omitted here because of the space considerations. Here, we present only a brief summary of sour simulation results.

Table 1 shows the results of the first experiment where an exhaustive search was made through all the possible plans for queries of size 4 relations. The results indicate that the G initially performs better (2nd row of table 1) and then H outperforms G.

%Queries	G	H
80%	0%	0%
85%	5%	15%
90%	22.5%	20%
95%	52.5%	50%

Table 1: 4 Rel's,128 P's,100 D's, Exhaustive

To validate the cost models for queries with more relations, doing an exhaustive search becomes quite time consuming as the experimentation was carried on a heavily loaded sparc 10. Hence random, sampling was used for queries with larger number of relations. The sampling was done with the requirement that at least $\sqrt{r}$ (where r is total number of plans in the execution space for the given query) samples are used during the simulations according to the *Monte-Carlo Techniques.*

Table 2 shows the results of next experiment where random sampling was used everything else remaining the same. The results show similar trends except they showed minor degradation in the error ranges. This probably is due to the fact that while sampling the plans, some of the best plans according to each individual estimator functions could have been missed. The result also shows that A (where A is the conventional *average work* cost function or the *Area* function used in sequential query optimization). A performs poorly compared to the G and H.

%Queries	G	H	A
80%	2.5%	0%	30%
85%	22%	11%	54%
90%	30%	25%	68%
95%	95%	70%	117%

Table 2: 4 Rel's,128 P's,100 D's,Random

Experiment 3 was conducted to see the effects of increasing the parallelism. It was done for a parallel machine with 1024 CPU's and 500 Disks. The results show similar curves for the G, H and A with respect to each other. The performance of the estimator functions improved massively compared to that of Experiment 2.

This indicates that as we scale up the machine, the heuristical functions performs better. The A estimator performed poorly again.

%Queries	G	H	A
80%	1%	0%	17%
85%	5%	2.5%	22.5%
90%	18%	12%	28%
95%	45%	20%	65%

Table 3: 4 Rel's,1024 P's,500 D's, Random

The Experiment 4 was conducted for a scaled down parallel machine (10 CPU's and 5 Disks). The curves show similar trends for G, H and A with respect to each other. The performance of the functions decreased as compared to Experiment 3. Thus indicating that the performance of the estimator functions is directly proportional to the parallelism of the machine. That is, they perform better with highly parallel machines.

%Queries	G	H	A
80%	2%	0%	30%
85%	17.5%	5%	50%
90%	47.5%	30%	70%
95%	105.5%	65%	195%

Table 4: 4 Rel's,10 P's,5 D's, Random

Experiment 5 was conducted to see the effects of increasing the dimensions of the query. The size of the queries was increased to 5 relations with a parallel machine with 128 CPU's and 100 Disks. Table 5 shows that the performance decreased for the error ranges 0% to 25%, but after that the estimator functions showed better performance than the 4 relation query set.

%Queries	G	H
80%	4%	10%
85%	22.5%	17.5%
90%	30%	26%
95%	45%	45%

Table 5: 5 Rel's,128 P's,100 D's, Random

Experiment 6 was conducted for the query set with 5 relations and a parallel machine with 1024 CPU's and 500 Disks. This experiment was performed to observe the effects of increasing the parallelism for the queries with dimension of 5 relations. The results again support the observation that the G and H perform better with increased parallelism.

%Queries	G	H	A
80%	5%	7.5%	65%
85%	10%	12.5%	75%
90%	27%	19%	100%
95%	46%	35%	141%

Table 6: 5 Rel's,1024 P's,500 D's, Random

Experiment 7 was conducted for the query set with 6 relations and a parallel machine with 128 CPU's and 100 Disks. This experiment was performed to observe the effects of increasing the dimensions of the queries. On comparison with results of experiment 2, the results again support the observation that the performance decreased for the 80 percentile row, but after that the performance was better than that of experiment 2. Similar trends are observed on comparing against experiment 5.

%Queries	G	H
80%	16.25%	16.25%
85%	18.75%	18.75%
90%	30%	28.4%
95%	52.5%	52.5%

Table 7: 6 Rel's,128 P's,100 D's, Random

The last experiment was done for a parallel machine of 1024 CPU's and 500 Disks with queries of dimension 8 relations. The results were simulated for a set of 23 queries. The experiment showed poor performance compared to the results of queries with 4 and 5 dimensions. The fact that the query set was very small for this case could have an effect on the poor performance of this experiment.

%Queries	G	H	A
80%	7.5%	22.5%	22.5%
85%	13%	28%	77.5%
90%	24%	54%	80%
95%	64%	64%	85%

Table 8: 8 Rel's,1024 P's,500 D's, Random

Consider a typical example. The H function consistently showed that 95% of the queries had error less than 20-60%. We take this to imply that if we optimized for the H function, then 95% of the time, we expect to be within 20-60% of the scheduled response time of queries.

Overall the H estimator performed the best followed closely by the G. Typically, we found that for 95%

of the queries, the optimal H plan had a performance that was within 1.2 to 1.6 times the performance of the optimal scheduled plan using the SCH function. For the G estimator function, we found that for 95% of the queries, the optimal G plan had a performance that was within 1.4 to 1.6 times the performance of the optimal scheduled plan.

The traditional *work* function neither performed well nor performed consistently. With one exception, we found that for 95% of the queries, the error ranged from 70% to 200%. The only exception was noticed in the case when the parallel machine was modeled to have 512 disks and 1024 processors, in which case the error was 65% for 95% of the queries (compared to 20% for the H).

It is clear that the accuracy of the traditional *work* cost function was not very good and worse, unreliable. It is also clear that the H function performed reasonably well when compared against a complex cost model, although not spectacularly well. Thus, if at most 20-60% degradation of performance is tolerable for 95% of the queries, then the G and the H cost models suffice. However, if more accuracy is needed, then we believe that new cost models that actually involve some form of scheduling would be needed. The design of more accurate cost models is beyond the scope of this paper and is left for future work. We do emphasize, that this is the first work that quantifies the errors of seemingly very reasonable cost models by comparing against accurate scheduling approaches.

7 Conclusions

The paper focuses on the design and validation of cost models for parallel query execution. From an efficiency perspective, we argue, that cost models, should not incorporate detailed schedulers, but instead estimate response time using estimator functions. Two important issues that arise in the design of estimator functions for response time are: (1) understanding the basis of the design of such functions, and (2) validating and measuring the accuracy of these estimator functions compared to a detailed scheduling function. Previous work on query optimization for parallel executions have not considered this problem.

We presented two cost functions G and H that carry minimal scheduling information. We presented theoretical arguments to show that G and H estimate response time to within a bounded factor of inaccuracy in non-trivial architectural scenarios.

We performed simulations to measure the accuracy of the estimator functions G and H as compared to the refined SCH function that incorporates a scheduling algorithm. We find that for 90-95% of the queries, all these functions have an inaccuracy ranging from 20-60%. We also tested the accuracy of the sequential work function, which is used as the objective function of centralized optimizers. We found that for 90-95% of the queries, the work cost function has an inaccuracy ranging from 70-150%. We conclude that G and H cost estimators are reasonable if the inaccuracy involved is tolerable. More research and/or performance evaluation is needed for the design of more accurate cost functions.

References

[BB90] K.P. Belkhale and P. Bannerjee. Approximate Algorithms for the Partitionable Independent Task Scheduling Problem, *International Conference on Parallel Processing*, 1990.

[DGS+] D.DeWitt, S. Ghandeharizadeh, D. Schneider, A. Bricker, H.Hsiao, R.Rasmussen. The Gamma Database Machine, *IEEE TKDE*, 1990.

[DeWGra92] D. DeWitt and J. Gray. The future of high performance database systems, *Communications of the ACM*, 1992.

[Gan92] S. Ganguly. Parallel Evaluation of Deductive Database Queries, *PhD thesis, University of Texas, Austin, 1992.*

[GHK92] S. Ganguly, W. Hasan and R. Krishnamurthy. Query Optimization for Parallel Executions, *SIGMOD*, 1992.

[GGMW] S. Ganguly, P. Gibbons, Y. Matias and A. Witkowski. AT&T Bell Labs Internal Technical Memorandum.

[GGJ78] M.R. Garey, R.L. Graham and D.S. Johnson. Performance Guarantees for Scheduling Algorithms, *Operations Research*, Jan. 1978.

[Goel95] A. Goel. Cost Models for Parallel Database Executions, *Masters' Thesis*, Department of Electrical Engineering, Rutgers University, New Brunswick, NJ, December 1994.

[Gra69] R.L. Graham. Bounds on Multiprocessing Timing Anomalies, SIAM J. Appl. Math., vol. 17, 1969.

[Gra66] R.L. Graham. Bounds on Multiprocessor Anomalies, *Bell System Technical Journal*, **45**, 1966.

[Hon91] W. Hong and M. Stonebraker. Optimization of Parallel Query Execution Plans in XPRS, *PDIS*, December 1991.

[LVZ93] R.S.G. Lanzelotte, P. Valduriez and M. Zait. On the Effectiveness of Optimization Search Strategies for Parallel Execution, *VLDB*, 1993.

[LST91] H Lu, M C. Shan and K.L. Tan. Optimization of Multi-Way Join Queries for Parallel Execution, *VLDB*, 1991.

[NSHL93] T H. Niccum, J. Srivastava, B. Himatsingka, J-Z. Li. A Tree-Decomposition Approach to the Parallel Execution of Relational Query Plans, *Technical Report, University of Minnesota at Minneapolis.*

[RSB94] S. Ramaswamy, S. Spatnekar and P. Banerjee. A Convex Programming Approach for Exploiting Data and Functional Parallelism on Distributed Memory Multicomputers. *International Conference on Parallel Processing, 1994.*

[Sch90] D. Schneider. Complex Query Processing in Multiprocessor Database Machines, *PhD thesis, University of Wisconsin, Madison, 1990.*

[SAC+] P. Selinger, M.M. Astrahan, D.D. Chamberlain, R.A. Lorie and T.G. Price. Access Path Selection in a Relational Database Management System, *SIGMOD*, 1979.

[SriEls93] Jaideep Srivastava and G. Elsesser. Query Optimization for Parallel Relational Databases, *PDIS*, 1993.

[SYT93] Eugene J. Shekita, Honesty C. Young and Kian-Lee Tan. Multi-Join Optimization for Symmetric Multiprocessors, *VLDB*, 1993.

[TL94] K-L. Tan, H. Lu. On resource scheduling of multi-join queries in parallel database systems, *Information Processing Letters*, 48 (1993).

[TWPY92] J. Turek, J.L. Wolf, K.R. Pattipati and P.S. Yu. Scheduling Parallelizable Tasks: Putting it All on the Shelf, *Sigmetrics*, 1992 .

[TWY92] J.W. Turek, J.L. Wolf and P.S. Yu. Approximate Algorithms for Scheduling Parallelizable Tasks, *Symposium on Parallel Algorithms and Architectures*, 1992.

[WC92] Q. Wang and K.H. Cheng. A Heuristic of Scheduling Parallel Tasks and its Analysis. *SIAM Journal on Computing*, April 1992.

[ZZBS94] M. Ziane, M. Zait, and P. Borla-Salamet. Parallel Query Processing in DBS3, *PDIS*, 1993.

Appendix

In this appendix, we present the cost formulae for data transfer from the processors executing node i to the processors executing node j under the assumption that each copy of operator o_i has Z_j exclusive buffers of size B bytes each, one buffer per destination processor. Once a buffer becomes full, it is transmitted to the destination. Let B_{ij} be the total number of bytes sent by processors executing node o_i to the processors executing node o_j. The following table summarizes the cost formulae that can be derived under this assumption.

All-to-all pattern

$$send_{ij} = \left\lceil \frac{B_{ij}}{B \cdot Z_i \cdot Z_j} \right\rceil \cdot \alpha \cdot Z_j + \frac{B_{ij}}{Z_i} \cdot \beta$$
$$receive_{ij} = \left\lceil \frac{B_{ij}}{B \cdot Z_i \cdot Z_j} \right\rceil \cdot \alpha \cdot Z_i + \frac{B_{ij}}{Z_j} \cdot \beta$$
$$delay_{ij} = B \cdot \delta$$

One-to-few pattern

$$send_{ij} = \left\lceil \frac{B_{ij}}{B \cdot max(Z_i, Z_j)} \right\rceil \cdot (max(Z_i, Z_j)/Z_i) \cdot \alpha + \frac{B_{ij}}{Z_i} \cdot \beta$$
$$receive_{ij} = \left\lceil \frac{B_{ij}}{B \cdot max(Z_i, Z_j)} i \right\rceil \cdot (max(Z_i, Z_j)/Z_j) \cdot \alpha + \frac{B_{ij}}{Z_j} \cdot \beta$$
$$delay_{ij} = B \cdot \delta$$

Let us derive the expression for $send_{ij}$ for the all-to-all pattern. By assumption of even distribution at the source, each processor has a total of B_{ij}/Z_i bytes that must be evenly distributed to each of the Z_j destination processor. Since the total number of bytes sent is B_{ij}/Z_i, the message component of the cost is $(B_{ij}/Z_i) \cdot \beta$. The number of bytes sent from one source to one destination processor is therefore $B_{ij}/(Z_i \cdot Z_j)$. Since messages are blocked in units of B, it follows that the number of messages sent between a source destination pair is $y = \lceil (B_{ij}/(B \times \cdot Z_i \cdot Z_j) \rceil$. Since each message has a startup cost of α, and there are Z_j destinations, the startup component of the cost is $y \cdot \alpha \cdot Z_j$. The $send_{ij}$ cost is the sum of the startup and message costs, which equals the expression as shown in the table. The rest of the entries can be derived in a similar fashion.