Scheduling Problems in Parallel Query Optimization

Chandra Chekuri*
Department of Computer Science
Stanford University
Stanford, CA 94305-2140

Waqar Hasan†
Department of Computer Science
Stanford University
Stanford, CA 94305-2140
and
Hewlett-Packard Laboratories

Rajeev Motwani‡
Department of Computer Science
Stanford University
Stanford, CA 94305-2140

Abstract

We introduce a class of novel multiprocessor scheduling problems that arise in the optimization of SQL queries for parallel machines. These consist of scheduling a tree of inter-dependent communicating operators while exploiting both inter-operator and intra-operator parallelism. We develop algorithms for the specific problem of scheduling a *Pipelined Operator Tree* in which all operators run in parallel using inter-operator parallelism. Weights associated with nodes and edges represent respectively the cost of operators and communication. Communication cost is incurred only if adjacent operators are assigned different processors. The optimization problem is to assign operators to processors so as to minimize the maximum processor load. We develop two approximation algorithms for this NP-hard problem. The faster algorithm has a performance ratio of 3.56 while the slower algorithm has a ratio of 2.87.

1 Introduction

Exploiting parallel execution [DG92, Val93] to speed up database queries presents a parallelism-communication trade-off. While work is divided among processors, the concomitant communication increases total work itself [Gra88, PMC+90]. We can represent the task to be scheduled as a *weighted operator tree* [Hon92, Sch90, HM94a] in which nodes represent atomic units of execution (operators) and directed edges represent the flow of data as well as timing constraints between operators. Weights associated with nodes and edges represent respectively the cost of operators and communication between them.

Scheduling a weighted operator tree on a parallel machine poses a class of novel multi-processor scheduling problems that differ from the classical [GLLK79] in several ways. Firstly, edges represent two kinds of timing constraints — parallel and precedence. Secondly, since data is transmitted in long streams, the important aspect of communication is the CPU overhead of sending/receiving messages and not the delay for signal propagation (see [PU87, PY88] for models of communication as delay). Thirdly, the set oriented nature of queries has led to intra-operator parallelism (relations are horizontally partitioned and a clone of the operator applied to each partition) in addition to inter-operator parallelism [DG92]. Finally, operators may have *distinct* weights since parallel database systems exploit *coarse-grained* parallelism.

We introduce several problems and focus on the specific problem of scheduling a *Pipelined Operator Tree* (POT scheduling). All edges in such a tree represent parallel constraints, i.e., all operators run in parallel. A schedule assigns operators to processors. Since edge weights represent the cost of remote communication, this cost is saved if adjacent operators share a processor. Given a schedule, the load on a processor is the sum of the weights of nodes assigned to it plus the weights of all edges that connect nodes on the processor to nodes on other processors. The response time (makespan) of a schedule is the maximum processor load. The optimization problem is to find a schedule with minimal response time.

POT scheduling is NP-hard since the special case in which all communication costs are zero is classical MULTIPROCESSOR SCHEDULING [GJ79]. We will measure the quality of algorithms by their *performance ratio* which is the ratio of the response time of the generated schedule to that of the optimal. Since the problem is intractable, our goal is to develop *approximation algorithms* that run in polynomial time and guarantee small bounds on the performance ratio [Mot92].

*E-mail: *chekuri@cs.stanford.edu*. Supported by an OTL grant and NSF Young Investigator Award CCR-9357849, with matching funds from IBM, Schlumberger Foundation, Shell Foundation, and Xerox Corporation.

†E-mail: *hasan@cs.stanford.edu*

‡E-mail: *rajeev@cs.stanford.edu*. Supported by an Alfred P. Sloan Research Fellowship, an IBM Faculty Development Award, an OTL grant, and NSF Young Investigator Award CCR-9357849, with matching funds from IBM, Schlumberger Foundation, Shell Foundation, and Xerox Corporation.

This paper develops two approximation algorithms. The faster algorithm has a tight performance ratio of 3.56 while the slower algorithm has a tight ratio of 2.87. Our earlier work [HM94a] developed algorithms for this problem that are known to have bounded performance ratios only when the shape of the tree is a path or a star.

In Section 2, we provide an overview of parallel query optimization and develop a model for scheduling problems. In Section 3, we review past work on the POT problem and describes our two-stage approach to the development of approximation algorithms for POT. In Section 4, we develop the LocalCuts algorithm and show it to have a performance ratio of 3.56. In Section 5, this algorithm is modified to yield the BoundedCuts algorithms which is shown to have a performance ratio of 2.87. Section 6 provides some initial results on managing partitioned parallelism and Section 7 discusses open problems.

2 A Model for Scheduling Problems

Figure 1 shows a two-phase approach [Hon92, Has95] for parallel query optimization. The first phase, JOQR (for Join Ordering and Query Rewrite), minimizes total cost and produces an annotated query tree that fixes aspects such as the order of joins and the strategy for computing each join. The second phase, *parallelization*, converts the annotated query tree into a parallel plan. Parallelization itself has two steps. The first converts the annotated query tree to an operator tree [GHK92, Hon92, Sch90]. The second schedules the operator tree on a parallel machine.

In this paper, we are only concerned with the second phase. Several approaches exist for the first phase; Hong and Stonebraker [HS91] used a conventional query optimizer while Hasan and Motwani [HM95] develop algorithms that incorporate communication costs.

We will first discuss the forms of available parallelism and how they are captured by the operator tree representation. We then describe how we model communication. Finally, we describe a variety of scheduling problems. The reader is referred to [HM94a, Has95] for more details.

2.1 Forms of Parallelism

Parallel database systems speed-up queries by exploiting *independent* and *pipelined* forms of inter-operator parallelism as well as intra-operator or *partitioned* parallelism. Independent parallelism simultaneously runs two operators with no dependence between them on distinct processors. Pipelined parallelism runs a consumer operator simultaneously with a producer operator on distinct processors. Partitioned parallelism uses several processors to run a single operator. It exploits the set-oriented nature of operators by partitioning the input data and running a copy of the operator on each processor.

2.2 Operator Trees

Available parallelism is represented as an operator tree $T = (V, E)$ with $V = \{1, \ldots, n\}$. Nodes represent operators. Functionally, an operator takes zero or more input sets and produces a single output set. Physically, it is a piece of code that is *deemed* to be atomic. Edges between operators represent the flow of data as well as timing constraints. As argued in our earlier work [HM94a], operators may be *designed* to ensure that any edge represents either a parallel or a precedence constraint.

Example 1 Figure 2 shows a query tree and the corresponding operator tree. Thin edges are pipelining edges, thick edges are blocking. A simple hash join is broken into `Build` and `Probe` operators. Since a hash table must be fully built before it can be probed, the edge from `Build` to `Probe` is blocking. A sort-merge join sorts both inputs and then merges the sorted streams. The merging is implemented by the `Merge` operator. In this example, we assume the right input of sort-merge to be pre-sorted. The operator tree shows the sort required for the left input broken into two operators `FormRuns` and `MergeRuns`. Since the merging of runs can start only after run formation, the edge from `FormRuns` to `MergeRuns` is blocking. ∎

The operator tree exposes the available parallelism. Partitioned parallelism may be used for any operator. Pipelined parallelism may be used between two operators connected by a pipelining edge. Two subtrees with no (transitive) timing constraints between them may run independently (eg: subtrees rooted at `FormRuns` and `Build`).

Definition 1 *A* pipelining *edge from operator i to j represents a parallel constraint that requires i and j to start at the same time and terminate at the same time. A* blocking *edge from i to j represents a* precedence constraint *that requires j to start after i terminates.*

A pipelining constraint is symmetric in i and j. The direction of the edge indicates the direction in which tuples flow but is immaterial for timing constraints. Since all operators in a pipelined subtree start and terminate simultaneously, heavier operators use a smaller fraction of the processor on which they run.

2.3 Model of Communication

The weight t_i of node i in an operator tree is the time to run the operator in isolation assuming all communication to be local. The weight c_{ij} of an edge from node i to j is the *additional* cpu overhead due to interprocessor communication. This overhead is incurred at

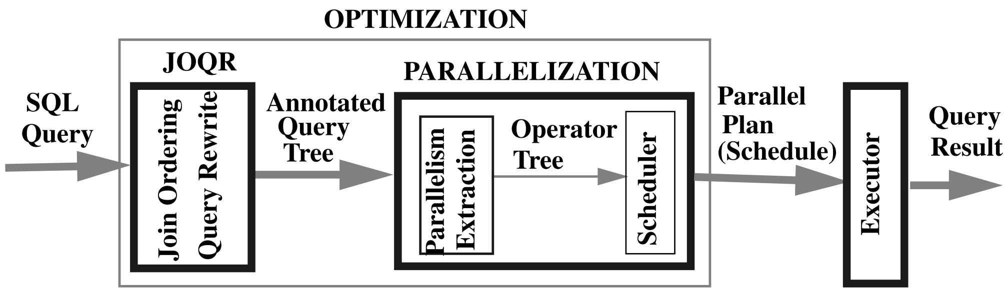

Figure 1: Parallel Query Processing:Software Architecture

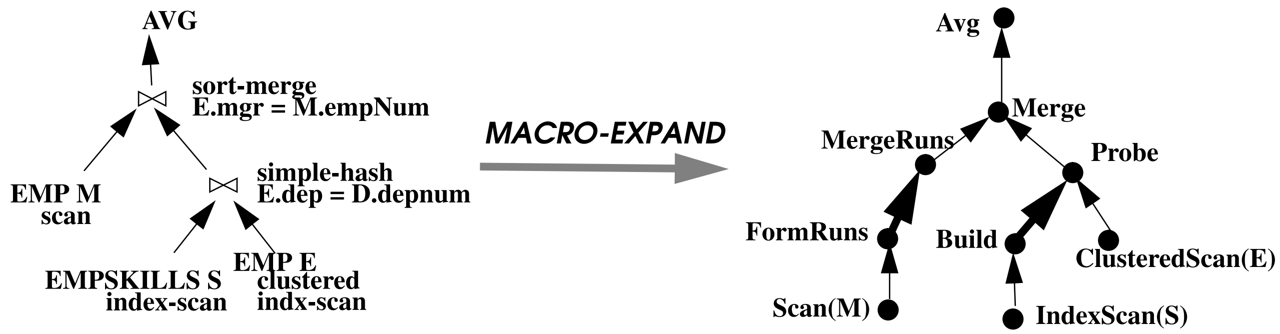

Figure 2: Macro-Expansion of Query Tree to Operator Tree (Parallelism Extraction)

both the sender and the receiver processor. A specific schedule incurs communication overheads only for the *fraction* of data that it actually communicates across processors. As discussed in [HM94a], conventional cost models (such as System R [SAC+79]) that estimate the sizes of the intermediate results may be easily adapted to estimate node and edge weights.

Figure 3 shows the extreme cases of communication costs of blocking and pipelining edges. Communication is saved when the two operators are on the same processor and totals to twice the edge weight when they are on distinct processors. For a blocking edge, communication occurs after the producer terminates and before the consumer starts. For a pipelined edge, communication is spread over the execution time of the entire operator. Note that since all operators in a pipeline start and terminate simultaneously, heavier operators consume a larger percentage of the processor they run on.

2.4 Scheduling Problems

We assume a parallel machine to consist of p identical processors. A schedule assigns operators to processors. We model partitioned parallelism as permitting processors to execute *fractions* of an operator. Depending on whether partitioned parallelism is allowed or not, the assignment of operators to processors is a fractional or 0-1 assignment. Since the goal of a parallel database system is to speedup queries, we are interested in finding schedules with minimal response time.

Definition 2 *The* response time *of a schedule is the elapsed time between starting query execution and fully producing the result.*

Figure 4 shows the division of the problem of scheduling operator trees into subproblems along two dimensions: the kinds of edges in an operator and whether schedules allows fractional assignment.

The next three sections investigate POT scheduling. Section 6 provides some initial results on POTP.

3 POT Scheduling: Our Approach

In this section we first review relevant definitions and results from our prior work [HM94a, HM94b]. We then describe a two-stage approach that we shall use in developing and analyzing our algorithms.

3.1 Problem Definition and Prior Results

Given a tree $T = (V, E)$ with positive real weights for each node and edge, the POT scheduling problem is to find a schedule with minimal response time. All edges in the tree are assumed to be pipelining edges. The following definitions make the problem precise.

Definition 3 *Given p processors and an operator tree $T = (V, E)$, a* schedule *is a partition of V, the set of nodes, into p sets $F_1, \ldots, F_p$ with set F_k allocated to processor k.*

Definition 4 *The load L_k on processor k is the cost of executing all nodes in F_k plus the overhead for*

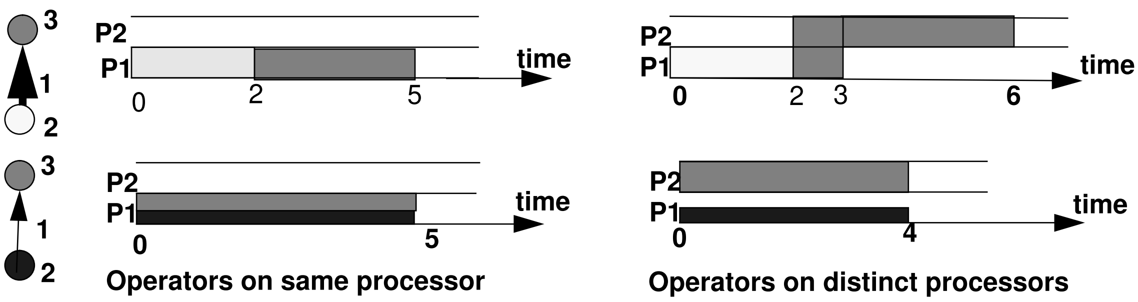

Figure 3: Communication Costs for Blocking and Pipelining Edges: Gantt Charts

	Partitioned Parallelism not used	Partitioned Parallelism used
Pipelining Edges Only	Pipelined Operator Tree Scheduling (POT)	Pipelined Operator Tree Scheduling with Partitioning (POTP)
Blocking Edges Only	Blocking Operator Tree Scheduling (BOT)	Blocking Operator Tree Scheduling with Partitioning (BOTP)
Pipelining & Blocking Edges	Operator Tree Scheduling (OT)	Operator Tree Scheduling with Partitioning (OTP)

Figure 4: Classification of Scheduling Problems

communicating with nodes on other processors, $L_k = \sum_{i \in F_k}[t_i + \sum_{j \notin F_k} c_{ij}]$.

Definition 5 *The response time of a schedule is the maximum processor load,* $L = \max_{1 \le k \le p} L_k$.

Example 2 Figure 5 shows a 2-processor schedule. Sets F_1 and F_2 are encircled, processor loads are underlined. F_1 results in a load of $L_1 = 31$ on processor 1 since the processor must pay for the three nodes in F_1 (5+5+10) as well as for the edges that connect to nodes on the other processor (5 + 6). Similarly $L_2 = 34$. The response time of the schedule is $L = \max(L_1, L_2) = 34$. We show edges as undirected since the parallel

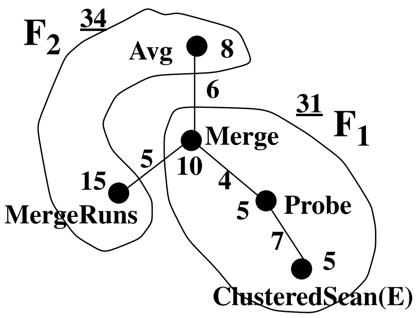

Figure 5: 2-processor Schedule

constraint represented by pipelining edges is symmetric. ∎

Our prior work [HM94a] showed that instead of scheduling a given tree, we may schedule the corresponding *monotone* tree. The monotone tree may be created by applying a linear-time pre-processing algorithm called GreedyChase to the given tree. Since GreedyChase simply collapses edges, a schedule for the original tree may be trivially obtained from a schedule for the monotone tree.

Monotone trees have the property that the weight of any node plus the weights of all incident edges is a lower bound on the response time of any schedule. This lower bound is useful in proving the performance ratios achieved by our algorithms.

Definition 6 *An operator tree is* monotone *if and only if any connected set of nodes, X, has a lower cost than any connected superset, Y, i.e., if $X \subset Y$ then cost$(X) <$ cost(Y).*

Lemma 1 *The response time of any schedule (independent of number of processors) for a monotone operator tree has a lower bound of* $\max_{i \in V}[t_i + \sum_{j \in V} c_{ij}]$.

Another interpretation of monotone trees is that they do not contain any *worthless* edges. These are edges whose communication cost is high enough to offset any benefits of using parallel execution for the two end points.

Definition 7 *An edge e_{ij} is* worthless *if and only if* $(c_{ij} \ge t_i + \sum_{k \ne j} c_{ik})$ *or* $(c_{ij} \ge t_j + \sum_{k \ne i} c_{jk})$.

The GreedyChase algorithm converts any tree into a monotone tree by repeatedly collapsing worthless edges. The following theorem shows that instead of scheduling a given tree, we may schedule the corresponding monotone tree.

Theorem 1 *Given p processors and an operator tree T with worthless edge (i, j), there exists an optimal schedule for T on p processors in which nodes i and j are assigned to the same processor.*

Example 3 In Figure 5, the edge between `Probe` and `ClusteredScan` is worthless since its weight exceeds the weight of `ClusteredScan`. The corresponding monotone tree is created by collapsing `Probe` and `ClusteredScan` into a single node of weight 10. ▮

In the remainder of the paper, we will assume operator trees to be monotone.

3.2 A Two-stage Approach

We divide the POT scheduling problem into two stages, *fragmentation* followed by *scheduling*. Fragmentation partitions the tree into *connected* fragments by *cutting* some edges. A cut edge is deleted from the tree. This should be interpreted as a decision to allocate the two end-points to distinct processors. The concomitant communication cost is captured by adding the weight of the deleted edge to the weights of both it's end-points. Any edge which is not cut is *collapsed*. This should be interpreted as a decision to schedule the two end-points on the same processor. Collapsing merges the two end-points into a single node which is assigned all the incident edges of the merged nodes and has weight equal to the sum of the weights of the merged nodes. We will view the result of fragmentation both as a set of fragments (some edges cut, collapsing implicit) as well as set of nodes (each edge cut or collapsed). Scheduling assigns the fragments produced by the first stage to processors.

The two stage approach offers conceptual simplicity and does not restrict the space of schedules. Any schedule defines a natural fragmentation corresponding to cutting exactly the inter-processor edges. For any given schedule, some scheduling algorithm will produce it from its natural fragmentation. Notice that the scheduling stage may assign two fragments that were connected by a cut edge to the same processor thus "undoing" the cutting. Thus, several fragmentations may produce the same schedule. In our analysis, we will ignore the decrease in communication cost caused by this implicit undoing of an edge cutting operation. This can only over-estimate the cost of our solution.

The two-stage approach allows us to use standard multiprocessor scheduling algorithms for the second stage. We will use the classical LPT [Gra69] algorithm. This is a greedy algorithm that assigns the largest unassigned job to the least loaded processor.

Given the use of LPT for scheduling, we may develop the conditions for a good fragmentation. There is an inherent tradeoff between total load and the weight of the heaviest connected fragment. If an edge is cut, communication cost is incurred thus increasing total load. If an edge is collapsed, a new node with a larger net weight is created, potentially increasing the weight of the largest connected fragment. Lemma 4 captures this trade-off and provides conditions on fragmentation for a bounded performance ratio. Before proceeding further, we make the following definitions.

Definition 8 *$R_i = t_i + \sum_j c_{ij}$ is the net weight of node i. $R = \max_i R_i$ is the maximum net weight over all nodes. $W = \sum_i t_i$ is the sum of the weights of all nodes. $\overline{W} = W/p$ is the average node weight per processor.*

Assuming fragmentation to produces q fragments with weights $M_1, \ldots, M_q$, we make the following definitions.

Definition 9 *$M = \max_i M_i$ is the weight of heaviest fragment. C is the the total communication cost incurred, which is twice the sum of the weights of the* cut *edges. $\overline{L} = (W+C)/p$ is the* average *load per processor.*

We use the subscript OPT to denote the same quantities for the natural fragmentation corresponding to an optimal schedule, for example, M_{OPT} for the weight of the heaviest fragment.

Example 4 Figure 6 shows the natural fragmentation for the schedule of Example 2. After the remaining edges are collapsed, we get three nodes with weights $M_1 = 14$, $M_2 = 20$, and $M_3 = 31$. Thus $M = \max\{M_1, M_2, M_3\} = 31$. $C = 22$ since the fragmentation cuts two edges with weights 5 and 6. Since the total node weight in the original tree is $W = 43$, we have $\overline{L} = (W + C)/p = (43 + 22)/2$. ▮

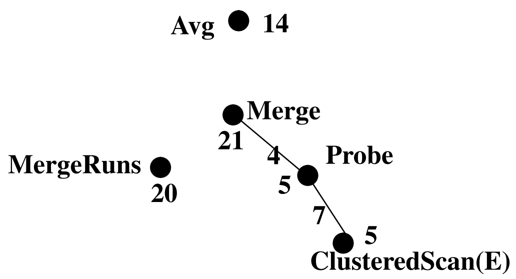

Figure 6: Fragments

The following two lemmas provide lower bounds on the value of the optimal solution.

Lemma 2 *$\overline{W} \leq \overline{L} \leq L$. In particular, $\overline{W} \leq \overline{L}_{\mathrm{OPT}} \leq L_{\mathrm{OPT}}$.*

Lemma 3 *$R \leq M \leq L$. In particular, $R \leq M_{\mathrm{OPT}} \leq L_{\mathrm{OPT}}$.*

In the following lemma, k_1 captures the effect of size of the largest fragment and k_2 the load increase due to communication.

Lemma 4 *Given a fragmentation with* $M \le k_1 L_{\mathrm{OPT}}$ *and* $\overline{L} \le k_2 \overline{L}_{\mathrm{OPT}}$*, scheduling using* LPT *yields a schedule with* $L/L_{\mathrm{OPT}} \le \max\{k_1, 2k_2\}$*.*

Proof: Let p_k be a *heaviest* loaded processor in an LPT schedule with response time L. Let M_j be the last fragment assigned to p_k. We will divide the analysis into two cases based on whether M_j is the only fragment on p_k or not.

If M_j is the only fragment on p_k, $L = M_j$ and by our assumption,

$$L = M_j \le M \le k_1 L_{\mathrm{OPT}}$$

Now consider the case when the number of fragments on p_k is at least 2. Since LPT assigns a job to the least loaded processor, the load on *any* processor must be at least $L - M_j$ when M_j was assigned to p_k. The total load $\sum_k L_k$ may be bounded as

$$\begin{aligned} & \sum_k L_k \ge (L - M_j)p + M_j \\ \Rightarrow \quad & L \le \frac{1}{p}\sum_k L_k + \left(1 - \frac{1}{p}\right) M_j \\ \Rightarrow \quad & L \le \overline{L} + M_j. \end{aligned}$$

Since LPT chooses the least loaded processor, the first p jobs are scheduled on distinct processors. Since there was at least one other fragment on p_k before M_j, there are at least $p+1$ fragments, each of them no lighter than M_j. Thus,

$$\begin{aligned} & \sum_k L_k \ge (p+1)M_j \\ \Rightarrow \quad & M_j \le \frac{1}{p+1}\sum_k L_k < \overline{L}. \end{aligned}$$

Combining the two inequalities shown above and using the assumption $\overline{L} \le k_2 \overline{L}_{\mathrm{OPT}}$, we obtain

$$\begin{aligned} L & \le \overline{L} + M_j \\ & \le 2\overline{L} \\ & \le 2k_2 \overline{L}_{\mathrm{OPT}}. \end{aligned}$$

Combining the two cases, we conclude $L/L_{\mathrm{OPT}} \le \max\{k_1, 2k_2\}$. ■

Using the above lemma, the best we can do is to find a fragmentation with $k_1 = k_2 = 1$ which would guaranteed a performance ratio of 2. Finding the best fragmentations is NP-complete in general. A *star* is a tree with only one non-leaf node. We show that even in this simple case, the problem is NP-complete.

Theorem 2 *Given a star* $T = (V, E)$*, bounds* B *and* C*, the problem of determining whether there is a partition of* V *such that no fragment is heavier than* B *and the total communication is no more than* C *is NP-complete.*

Proof Sketch: We reduce the classical *knapsack* problem [GJ79] to the above problem. Let an instance of the knapsack problem be specified by a bag size S and n pairs (w_i, p_i) where each pair corresponds to an object of weight w_i with profit p_i. We can assume without loss of generality that $p_i \le w_i$ for all i since all p_i can be scaled. Consider a star T with $n+1$ nodes obtained from the knapsack instance. We label the nodes of T from 0 to n with the center as 0. We set $c_{i0} = p_i/2$ and $t_i = w_i + c_i$ and $B = S + \sum_i c_i$. We claim that the minimum communication cost for the star instance is C if and only if the maximum profit for the knapsack instance is $\sum_i p_i - C$. ■

We remark that the problem is polynomially solvable when the tree is restricted to be a path. A path is a tree with exactly two leaves.

The next two section focus on algorithms to find good fragmentations that guarantee low values for k_1 and k_2.

4 The LocalCuts Algorithm

We now develop a linear time algorithm for fragmentation called LocalCuts. We show bounds on the weight of the heaviest fragment as well as on the load increase due to communication. Application of Lemma 4 shows the algorithm to have a performance ratio of 3.56.

LocalCuts repeatedly picks a leaf and determines whether to cut or collapse the edge to its parent. It makes the decision based on *local* information, the ratio of the leaf weight to the weight of the edge to its parent. The basic intuition is that if the ratio is low, then collapsing the edge will not substantially increase the *net* weight of the parent. If the ratio is high, the communication cost incurred by cutting will be relatively low and can be amortized to the weight of the node cut off. One complication is that cutting or collapsing an edge changes node weights. Our analysis *amortizes* the cost of cutting an edge over the total weight of all nodes that were collapsed to produce the leaf.

In the following discussion we assume that the tree T has been *rooted* at some arbitrary vertex. We will refer to the fragment containing the root as the *residual tree.* A *mother* node in a rooted tree is a node all of whose children are leaves. The algorithm uses a parameter $\alpha > 1$. We will later show (Theorem 3) how this parameter may be chosen to minimize the performance ratio.

The LocalCuts Algorithm:
Input: Operator tree T, parameter $\alpha > 1$.
Output: Partition of T into fragments $F_1, \ldots, F_k$.

while there is a mother node m with a child j **do**
  if $t_j > \alpha c_{jm}$ **then** cut e_{jm}
  else collapse e_{jm}.

return fragments induced by the cut edges in T.

The running time of the LocalCuts algorithm is $O(n)$. The following lemma shows a bound on the weight of the resulting fragments.

Lemma 5 *Any fragment produced by* LocalCuts *has weight less than αR, which implies $M < \alpha R$.*

Proof: Consider an arbitrary fragment produced in the course of the algorithm. Let m be the *highest* level node in the fragment, with children $1, \ldots, d$. The node m is picked as a mother node at some stage of the algorithm. Now, $R_m = c_{mp} + t_m + c_{m1} + \ldots + c_{md}$ where c_{mp} is the weight of the edge from m to it's parent. Collapsing child j into m, corresponds to replacing c_{mj} by t_j. Since the condition for collapsing is $t_j < \alpha c_{mj}$, collapsing children can increase R_m to at most αR_m which is no greater than αR. ∎

We now use an amortization argument to show that the communication cost incurred by the LocalCuts algorithm is bounded by a constant factor of the total node weight, W.

Lemma 6 *The total communication cost of the partition produced by the* LocalCuts *algorithm is bounded by $\frac{2}{\alpha-1}W$, that is $C \leq \frac{2}{\alpha-1}W$.*

Proof: We associate a *credit* p_i with each node i and *credit* p_{jk} with each edge e_{jk}. Initially, edges have zero credit and the credit of a node equals it's weight; thus, the total initial credit is W. The total credit will be conserved as the algorithm proceeds. When a node is cut or collapsed, it's credit is taken away and either transferred to another node or to an edge that is cut. The proof is based on showing that when the algorithm terminates, every edge that is cut has a credit equal to $(\alpha - 1)$ times it's weight. This allows us to conclude that the total weight of the cut edges is bounded by $W/(\alpha - 1)$. This would then imply that $C \leq \frac{2}{\alpha-1}W$. We abuse notation by using t_i for the *current* weight of a node in the residual tree. We now prove the following invariants using an inductive argument.

1. Each node has a credit greater than or equal to it's *current* weight in the residual tree, i.e., $p_i \geq t_i$.
2. Each *cut* edge e_{im} has a credit equal to $(\alpha - 1)$ times it's weight, i.e., $p_{im} = (\alpha - 1)c_{im}$.

As the base case, these invariants are trivially true at the beginning of the algorithm. As the inductive step, suppose these invariants are true up to k iterations and we consider leaf node j with mother m in the $(k+1)$st iteration. If j is collapsed, $t_m^{\mathrm{NEW}} = t_m + t_j$. We use the superscript NEW to indicate the values at the next iteration. By transferring the credit of j to m, we get $p_m^{\mathrm{NEW}} = p_j + p_m$. Since $p_j \geq t_j$ and $p_m \geq t_m$, by the inductive hypothesis we have $p_m^{\mathrm{NEW}} \geq t_m^{\mathrm{NEW}}$ and both invariants are preserved.

If j is cut, $t_m^{\mathrm{NEW}} = t_m + c_{jm}$. We need to transfer a credit of c_{jm} to m to maintain the first invariant. The remaining credit $p_j - c_{jm}$ may be transferred to the edge e_{jm}. By the induction hypothesis, we have $p_j - c_{jm} \geq t_j - c_{jm}$ and since edge e_{jm} was cut, $p_j - c_{jm} > (\alpha - 1)c_{jm}$. Thus sufficient credit is available for the second invariant as well. ∎

The previous two lemmas combined with Lemma 4, allow us to bound the performance ratio guaranteed by LocalCuts. The following theorem states the precise result and provides a value for the parameter α.

Theorem 3 *Using* LPT *to schedule the fragments produced by* LocalCuts *with $\alpha = (3 + \sqrt{17})/2$ gives a performance ratio of $(3 + \sqrt{17})/2 \sim 3.56$.*

Proof: From Lemma 6 and Lemma 2,

$$\overline{L} = \frac{W + C}{p} \leq \frac{\alpha + 1}{\alpha - 1}\overline{W} \leq \frac{\alpha + 1}{\alpha - 1}\overline{L}_{\mathrm{OPT}}.$$

Combining this with Lemma 5 and using Lemma 4 we conclude

$$\frac{L}{L_{\mathrm{OPT}}} \leq \max\left\{\alpha, \frac{2(\alpha + 1)}{\alpha - 1}\right\}$$

Observing that the max is minimized when $\alpha = 2(\alpha + 1)/(\alpha - 1)$, we obtain $\alpha = (3 + \sqrt{17})/2$ and $L/L_{\mathrm{OPT}} \leq (3 + \sqrt{17})/2$. ∎

The performance ratio of LocalCuts is tight. Consider a star in which the center node with weight δ is connected by edges of weight 1 to $n - 1$ leaves, each of weight $\alpha = 3.56$. Suppose the star is scheduled on $p = n$ processors. LocalCuts will collapse all leaves and produce a single fragment of weight $(n - 1)\alpha + \delta$. The optimal schedule consists of cutting all edges to produce $n - 1$ fragments of weight $1 + \alpha$ and one fragment of weight $n - 1 + \delta$. When $n > 5$, the performance ratio is $((n - 1)\alpha + \delta)/(n - 1 + \delta)$ which approaches α as δ goes to zero.

5 The BoundedCuts Algorithm

The LocalCuts algorithm determines whether to collapse a leaf into its mother based on the ratio of the leaf weight to the weight of the edge to its mother. The decision is independent of the current weight of the mother node. From the analysis of LocalCuts, we see that the weight of the largest fragment is bounded by αR_m, where m is the highest level node in the fragment (Lemma 5). If R_m is small compared to M_{OPT}, we may cut expensive edges needlessly. Using a bound that is independent of R_m should reduce communication costs.

The analysis of LocalCuts showed the trade-off between total communication ($C \leq \frac{2}{\alpha-1}W$) and the bound on fragment size ($M < \alpha R$). Reduced communication should allow us to afford a lower value of α, thus reducing the largest fragment size and the performance ratio.

We now discuss a modified algorithm called BoundedCuts that uses a uniform bound B at each mother node. It also cuts off light edges in a manner similar to LocalCuts. Our analysis will show that the modified algorithm improves the performance ratio to 2.87. We will show the ratio to be tight. Our analysis of communication costs uses lower bounds on C_{OPT}, the communication incurred in some fixed optimal schedule.

The algorithm below is stated in terms of three parameters α, β and B that are assumed to satisfy $\beta \geq \alpha > 1$, and $M_{\mathrm{OPT}} \leq B \leq L_{\mathrm{OPT}}$. Our analysis uses these conditions and we shall later show how the values of these parameters may be fixed.

The BoundedCuts Algorithm:
Input: Operator tree T, real parameters α, β, and B, where $\beta \geq \alpha > 1$ and $B \geq R$.
Output: Partition of T into connected fragments $T_1, \ldots, T_k$.

1. **while** there exists a mother node m **do**
- **2.** partition children of m into sets N_1, N_2 such that child $j \in N_1$ if and only if $t_j / c_{mj} \geq \beta$;
- **3.** cut e_{mj} for $j \in N_1$; **(β rule)**
- **4.** **if** $R_m + \sum_{j \in N_2}(t_j - c_{mj}) \leq \alpha B$
  - **5.** **then** collapse e_{mj} for all $j \in N_2$
  - **6.** **else** cut e_{mj} for all $j \in N_2$; **(α rule)**

7. **return** resulting fragments $T_1, \ldots, T_k$.

Lemma 7 *Any fragment produced by* BoundedCuts *has weight at most* αB. *As a consequence,* $M \leq \alpha L_{\mathrm{OPT}}$.

Proof: Since the weight of a fragment increases only when some edge is collapsed, the explicit check in line 4 ensures the lemma. ■

Let $\mathcal{C}$ denote the set of edges cut by BoundedCuts. We cut edges using two rules, the β rule in Step 3 and the α rule in Step 6. Let $\mathcal{C}_\beta$ and $\mathcal{C}_\alpha$ denote the edges cut using the respective rules. $\mathcal{C}_\beta$ and $\mathcal{C}_\alpha$ are disjoint and $\mathcal{C}_\beta \cup \mathcal{C}_\alpha = \mathcal{C}$. Let C_β and C_α denote the communication

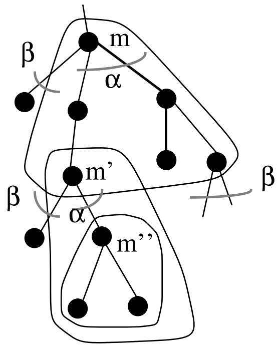

Figure 7: Subtrees T_m, $T_{m'}$, $T_{m''}$ for nodes m, m', m''

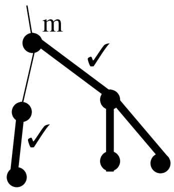

Figure 8: $\mathcal{C}^m_{\mathrm{OPT}}$

cost incurred due to edges in $\mathcal{C}_\beta$ and $\mathcal{C}_\alpha$ respectively. We bound C_β and C_α in Lemmas 8 and 10.

Lemma 8 $C_\alpha \leq \dfrac{\beta - 1}{\alpha - 1} C_{\mathrm{OPT}}$.

The proof of the lemma requires several definitions and lemmas.

Definition 10 *Let* $T_i = (V_i, E_i)$ *denote a subtree of* $T = (V, E)$ *rooted at* i, *defined as follows:* V_i *includes* i, *children of* i *that are not cut off by the* β *rule, and all nodes that eventually collapse into a child of* i; E_i *consists of all edges* $e_{kj} \in E$ *such that* $k, j \in V_i$. *The weight of an edge in* E_i *is the same as the corresponding edge in* E. *The weight of node* $j \in V_i$ *is the weight of* j *in* T *plus the weights of all incident edges that are not in* E_i, *i.e.,* $t_j^{(T_i)} = t_j^{(T)} + \sum_{\ell \in V - V_i} c_{j\ell}$.

Figure 7 illustrates the definition of T_i. With respect to the figure, the weight of m in T_m equals the original weight plus the weight of the two edges that connect m to nodes not in T_m.

Definition 11 W_i *is the total weight of all nodes in* T_i.

Definition 12 $\mathcal{C}_{\mathrm{OPT}}$ *is defined to be the set of edges in tree* T *that are cut in a fixed optimal solution.*

Definition 13 $\mathcal{C}^i_{\mathrm{OPT}}$ *is set of edges formed by starting with the edges* $\mathcal{C}_{\mathrm{OPT}} \cap E_i$ *and deleting all edges* e_{kj} *for which there exists* $e_{m\ell} \in \mathcal{C}_{\mathrm{OPT}} \cap E_i$ *with* m *being an ancestor of* k.

$\mathcal{C}^i_{\mathrm{OPT}}$ is a subset of the edges of T_i that are cut in the optimal. Figure 8 shows the edges in T_m that are cut by a fixed optimal schedule as thick edges. The subset of edges that forms $\mathcal{C}^m_{\mathrm{OPT}}$ are checked off.

Definition 14 *$\mathcal{C}^i_\alpha$ is defined to be the set of edges in tree T_i that are cut by the α rule. C^i_α is the total weight of the edges in $\mathcal{C}^i_\alpha$.*

Lemma 9 *If m and m' are distinct mother nodes where we cut using the α rule, then $\mathcal{C}^m_{\mathrm{OPT}} \cap \mathcal{C}^{m'}_{\mathrm{OPT}} = \phi$ and $\mathcal{C}^m_\alpha \cap \mathcal{C}^{m'}_\alpha = \phi$.*

Proof: The lemma follows since, by their definition, trees T_m and $T_{m'}$ do not share any edges (see Figure 7). ■

Proof of Lemma 8: By Lemma 9, it suffices to establish

$$C^m_\alpha \le \frac{\beta - 1}{\alpha - 1} C^m_{\mathrm{OPT}}$$

for each mother node m where we use the α rule to cut edges. Let the set $\mathcal{C}^m_{\mathrm{OPT}}$ consist of s edges $e_{m_1 z_1}, \ldots, e_{m_s z_s}$. These edges partition T_m into $s+1$ fragments. From the definition of $\mathcal{C}^m_{\mathrm{OPT}}$ it follows that one fragment, F_m, contains nodes m and $m_1, \ldots, m_s$ (some of these may be the same as m). Let the remaining fragments be $F_1, \ldots, F_s$, with F_j containing node z_j. We have

$$C^m_{\mathrm{OPT}} = \sum_{1 \le j \le s} c_{m_j z_j}$$

Since no fragment in the optimal is larger than M_{OPT}, the total node weight in fragment F_m is at most $M_{\mathrm{OPT}} - \sum_{1 \le j \le s} c_{m_j z_j}$. Thus, letting Q_j be the total node weight in fragment F_j for $j = 1, \ldots, s$, we have

$$M_{\mathrm{OPT}} - \sum_{1 \le j \le s} c_{m_j z_j} + \sum_{1 \le j \le s} Q_j \ge W_m.$$

We applied the α rule at m. Since children cut by the α rule are in T_m, $W_m > \alpha B$. Since $B \ge M_{\mathrm{OPT}}$, we have $W_m > \alpha M_{\mathrm{OPT}}$ which reduces the above equation to:

$$\sum_{1 \le j \le s} (Q_j - c_{m_j z_j}) > (\alpha - 1) M_{\mathrm{OPT}}$$

Since no edge in T_m was cut by the β rule, we must have $Q_j < \beta c_{m_j z_j}$ which results in

$$\sum_{1 \le j \le s} (\beta - 1) c_{m_j z_j} > (\alpha - 1) M_{\mathrm{OPT}}$$

$$\Rightarrow \quad M_{\mathrm{OPT}} < \frac{\beta - 1}{\alpha - 1} \sum_{1 \le j \le s} c_{m_j z_j} = \frac{\beta - 1}{\alpha - 1} C^m_{\mathrm{OPT}}.$$

Since $C^m_\alpha < R_m \le M_{\mathrm{OPT}}$, we have the desired result:

$$C^m_\alpha \le \frac{\beta - 1}{\alpha - 1} C^m_{\mathrm{OPT}}$$

■

Using techniques similar to those in the proof of Lemma 6, we show the following bound on C_β.

Lemma 10 $\mathcal{C}_\beta \le \dfrac{2}{\beta - 1} W - \dfrac{\alpha - 1}{\beta - 1} \mathcal{C}_\alpha$.

Proof: We use a credit based argument similar to that of Lemma 6. For each edge in $\mathcal{C}_\beta$ we associate a credit of $(\beta - 1)$ times it's weight and for each $\mathcal{C}_\alpha$ edge we maintain a credit of $(\alpha - 1)$ times it's weight. The proof for $\mathcal{C}_\beta$ edges is similar to that in Lemma 6. For $\mathcal{C}_\alpha$ edges, we cannot use a similar argument since the weight of the leaf being cut off, is not necessarily α times the weight of the edge to it's parent. But consider all the edges cut off at a mother node. From the algorithm we have $R_m + \sum_{j \in N_2} (t_j - c_{mj}) > \alpha B$. From this we see that even though each leaf is not heavy enough, the combined weight of all the leaves being cut off at a mother node is sufficient for a credit of $(\alpha - 1)$ times the weight of the edges cut. Since we start with an initial credit of W, the result follows. ■

Combining Lemmas 8 and 10, we obtain the following.

Lemma 11 $C = C_\beta + C_\alpha \le \dfrac{2}{\beta - 1} W + \dfrac{\beta - \alpha}{\alpha - 1} C_{\mathrm{OPT}}$.

We need the following technical lemma before we prove the main theorem.

Lemma 12 *For $\beta \ge \alpha > 1$, the function*

$$m(\alpha, \beta) = \max \left\{ \alpha, \frac{2(\beta + 1)}{\beta - 1}, \frac{2(\beta - \alpha)}{\alpha - 1} \right\}$$

is minimized when

$$\alpha = \frac{2(\beta + 1)}{\beta - 1} = \frac{2(\beta - \alpha)}{\alpha - 1}$$

The minimum value is 2.87 when $\alpha \sim 2.87$ and $\beta \sim 5.57$.

Proof: We observe that $f(\alpha, \beta) = \alpha$ is strictly increasing in α, $h(\alpha, \beta) = 2(\beta - \alpha)/(\alpha - 1)$ is strictly decreasing in α, $g(\alpha, \beta) = 2(\beta + 1)/(\beta - 1)$ is strictly decreasing in β, and h is strictly increasing in β. From this it is easy to verify that at the optimum point, both f and g must be equal to the optimum value. If either them is not the max-value of the max, then appropriately change α/β to make this happen, and note that this can only reduce the value of h. From this it follows that all three terms are equal at the optimum. Eliminating β from the above two equations gives us

$$\alpha^3 - \alpha^2 - 4\alpha - 4 = 0$$

which on solving yields the claimed values for α, β and the minimum. ■

Theorem 4 *Using* LPT *to schedule the fragments produced by* BoundedCuts *with* $\alpha = 2.87$, *and* $\beta = 5.57$ *gives a performance ratio of* 2.87.

Proof: Using Lemma 11, we have

$$\begin{aligned} \overline{L} = \frac{W+C}{p} &\leq W + \frac{2}{\beta-1}W + \frac{\beta-\alpha}{\alpha-1}C_{\mathrm{OPT}} \\ &\leq \max\left\{\frac{\beta+1}{\beta-1}, \frac{\beta-\alpha}{\alpha-1}\right\} \times \overline{L}_{\mathrm{OPT}}. \end{aligned}$$

Using the bound on $\overline{L}$ from the above equation and from the bound on M from Lemma 7, we can apply Lemma 4 to obtain

$$\begin{aligned} \frac{L}{L_{\mathrm{OPT}}} &\leq \max\left\{\alpha, 2\left(\max\left\{\frac{\beta+1}{\beta-1}, \frac{\beta-\alpha}{\alpha-1}\right\}\right)\right\} \\ &\leq \max\left\{\alpha, \frac{2(\beta+1)}{\beta-1}, \frac{2(\beta-\alpha)}{\alpha-1}\right\}. \end{aligned}$$

From Lemma 12, the right hand side of the above inequality is minimized at the values stated in the theorem, and this shows that $L/L_{\mathrm{OPT}} \leq 2.87$. ∎

The performance ratio of BoundedCuts is tight. The example is similar to that for LocalCuts i.e. a star in which the center node with weight δ is connected by edges of weight 1 to $n-1$ leaves each of weight $\alpha = 2.87$. Suppose the star is scheduled on $p = n$ processors. The optimal schedule consists of cutting all edges to produce $n-1$ fragments of weight $1+\alpha$ and one fragment of weight $n-1+\delta$. Taking $n > 4$, $M_{\mathrm{OPT}} = L_{\mathrm{OPT}} = n-1+\delta$. BoundedCuts will collapse all leaves and produce a single fragment of weight $(n-1)\alpha+\delta$ (since $B = L_{\mathrm{OPT}}$, this does not exceed αB). The performance performance ratio is therefore $((n-1)\alpha+\delta)/(n-1+\delta)$ which approaches α as δ goes to zero.

The results in this section rely on the fact that the bound B used in BoundedCuts satisfies $M_{\mathrm{OPT}} \leq B \leq L_{\mathrm{OPT}}$. Since we do not know the optimal partition, we do not know M_{OPT} or L_{OPT}. However, we can ensure that we try a value of B that is as close as we want to L_{OPT}. The following theorem makes the idea more precise.

Theorem 5 *For any* $\epsilon > 0$, *we can ensure that we run* BoundedCuts *with a bound* B *satisfying* $L_{\mathrm{OPT}} \leq B \leq (1+\epsilon)L_{\mathrm{OPT}}$. *This yields a performance ratio of* $(1+\epsilon)2.87$ *with a running time of* $O(\epsilon^{-1}np\log n)$.

Proof: From Lemmas 2 and 3, $\max\{\overline{W}, R\}$ is a lower bound on L_{OPT}. W is an upper bound since we can always schedule the entire tree on a single processor. Thus, $\overline{W} \leq L_{\mathrm{OPT}} \leq p\overline{W}$. We can try the value $B = \epsilon k\overline{W}$ for each integer k satisfying $1/\epsilon \leq k \leq p/\epsilon$. For each such value, we run BoundedCuts followed by LPT and take the best schedule. This guarantees that we will use a bound $L_{\mathrm{OPT}} \leq B \leq (1+\epsilon)L_{\mathrm{OPT}}$. From the previous analysis, if we use such a bound, we get a performance ratio of $(1+\epsilon)2.87$. There are $(p-1)/\epsilon$ values for k, LPT requires $O(n\log n)$ time, and BoundedCuts requires $O(n)$. Thus the total time for all values of B is $O(\epsilon^{-1}np\log n)$. ∎

6 Exploiting Partitioned Parallelism

We observe that when partitioned (i.e intra-operator) parallelism is allowed, and all communication costs are zero, it is easy to find the optimal schedule.

Lemma 13 *The* symmetric schedule *that equally partitions each operator over all processors is an optimal schedule for an operator tree with both pipelined and blocking edges provided all edge weights are zero.*

However, when communication is non-zero, the tradeoff between parallelism and communication presents a challenge. We now mention some results on the POTP problem that extends POT by allowing partitioned parallelism. Since the work for a single operator may be partitioned among several processors, a schedule is a matrix A in which entry a_{ik} is the fraction of operator i allocated to processor k.

Definition 15 *A* schedule *(or* processor allocation matrix*) is a* $n \times p$ *matrix* A *with entries* $a_{ik} \geq 0$ *such that* $\sum_{1\leq k\leq p} a_{ik} = 1$.

Suppose operator i produces a data stream consumed by operator j. Given schedule A, fraction a_{ik} of the data stream will be produced on processor k. Of this, fraction a_{jk} is consumed by the local clone of j and fraction $1-a_{jk}$ by non-local clones. Thus, on processor k, operator i incurs a communication cost of $c_{ij}a_{ik}(1-a_{jk})$ with operator j. Generalizing, the total communication cost (with all other operators) incurred by i on processor k is $\sum_{1\leq j\leq n} a_{ik}(1-a_{jk})c_{ij}$. The total cost of operator i on processor k is $a_{ik}t_i + \sum_{1\leq j\leq n} a_{ik}c_{ij}(1-a_{jk})$.

Definition 16 *The load* $L_k(A)$ *on processor* k *in schedule* A, *is given by*

$$L_k(A) = \sum_{1\leq i\leq n} a_{ik}t_i + \sum_{1\leq i,j\leq n} a_{ik}(1-a_{jk})c_{ij}.$$

The response time *of* A *is* $\max_{1\leq k\leq p} L_k(A)$.

POTP is a non-linear optimization problem. The presence of *max* in the definition of response time means that it is not a quadratic programming problem. At this time we can show the following results:

Theorem 6 *POTP is NP-hard.*

Theorem 7 *For a tree with two nodes scheduled on any number of processors, either the symmetric schedule or the schedule that places both nodes on the same processor is optimal.*

Theorem 8 *For instances of POTP for which $R_{max} \leq \overline{W}$, using* LocalCuts *for fragmentation followed by* LPT *for scheduling gives a performance ratio of 3.56.*

7 Future Work

Parallel query optimization poses a variety of novel scheduling problems. We have developed algorithms for the POT problem and have some initial results on the POTP problem that permits the use of partitioned parallelism.

The problems that disallow partitioned parallelism are NP-hard even for the case of zero communication costs (see also [Ull75]). The BOT problem with zero communication costs is NP-hard since multiprocessor scheduling may be reduced to BOT by constructing a star whose leaves are the jobs in multi-processor scheduling. The leaves are all connected by blocking edges to the root that represents a job with zero cost.

The practical importance of the problems presented in this paper rests on the premise that communication is a significant component of the cost of processing a query in parallel. The reader is referred to Pirahesh et. al. [PMC+90], Gray [Gra88] and Hasan et. al. [HM95] for such evidence.

The parallelism-communication trade-off is not the only concern in parallel query optimization. We have assumed that a parallel machine consists of a set of processors that communicate over an inter-connect. Enhancing the machine model to incorporate disks and memories presents challenging problems. Hong [Hon92] develops a method for balancing CPU and disk while ignoring communication costs. Incorporating memory as a separate resource into optimization algorithms for complex queries has been insufficiently addressed even for uniprocessor machines.

Acknowledgements

Thanks are due to Jim Gray, Jeff Ullman and Gio Wiederhold for useful discussions.

References

[DG92] D. J. DeWitt and J. Gray. Parallel Database Systems: The Future of High Performance Database Systems. *Communications of the ACM*, 35(6):85–98, June 1992.

[GHK92] S. Ganguly, W. Hasan, and R. Krishnamurthy. Query Optimization for Parallel Execution. In *Proceedings of ACM-SIGMOD International Conference on Management of Data*, June 1992.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability.* W.H. Freeman and Company, 1979.

[GLLK79] R.L Graham, E.L. Lawler, J.K. Lenstra, and A.H.G Rinnooy Kan. Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[Gra69] R.L. Graham. Bounds on Multiprocessing Timing Anomalies. *SIAM Journal of Applied Mathematics*, 17(2):416–429, March 1969.

[Gra88] J. Gray. The Cost of Messages. In *Proceedings of the Seventh ACM Symposium on Principles of Distributed Computing*, pages 1–7, Toronto, Ontario, Canada, August 1988.

[Has95] W. Hasan. *Optimizing Response Time of Relational Queries by Exploiting Parallel Execution.* PhD thesis, Stanford University, 1995. In preparation.

[HM94a] W. Hasan and R. Motwani. Optimization Algorithms for Exploiting the Parallelism-Communication Tradeoff in Pipelined Parallelism. In *Proceedings of the Twentieth International Conference on Very Large Data Bases*, pages 36–47, Santiago, Chile, September 1994.

[HM94b] W. Hasan and R. Motwani. Optimization Algorithms for Pipelined Parallelism. Technical report, HP Laboratories, 1994. HPL-94-50.

[HM95] W. Hasan and R. Motwani. Coloring Away Communication in Parallel Query Optimization, 1995. Submitted for publication.

[Hon92] W. Hong. *Parallel Query Processing Using Shared Memory Multiprocessors and Disk Arrays.* PhD thesis, University of California, Berkeley, August 1992.

[HS91] W. Hong and M. Stonebraker. Optimization of Parallel Query Execution Plans in XPRS. In *Proceedings of the First International Conference on Parallel and Distributed Information Systems*, December 1991.

[Mot92] R. Motwani. *Lecture Notes on Approximation Algorithms (Volume I).* Report No. STAN-CS-92-1435, Department of Computer Science, Stanford University, June 1992.

[PMC+90] H. Pirahesh, C. Mohan, J. Cheung, T.S. Liu, and P. Selinger. Parallelism in Relational Data Base Systems: Architectural Issues and Design Approaches. In *Second International Symposium on Databases in Parallel and Distributed Systems*, Dublin, Ireland, 1990.

[PU87] C.H. Papadimitriou and J.D. Ullman. A Communication-Time Tradeoff. *SIAM Journal of Computing*, 16(4):639–646, August 1987.

[PY88] C.H. Papadimitriou and M. Yannakakis. Towards an Architecture-Independent Analysis of Parallel Algorithms. In *Symposium on Theory of Computing*, pages 510–513, 1988.

[SAC+79] P. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access Path Selection in a Relational Database Management System. In *Proceedings of ACM-SIGMOD International Conference on Management of Data*, 1979.

[Sch90] D. A. Schneider. *Complex Query Processing in Multiprocessor Database Machines.* PhD thesis, University of Wisconsin—Madison, September 1990. Computer Sciences Technical Report 965.

[Ull75] J.D. Ullman. NP-Complete Scheduling Problems. *JCSS*, 10:384–393, 1975.

[Val93] P. Valduriez. Parallel Database Systems: Open Problems and New Issues. *Distributed and Parallel Databases: An International Journal*, 1(2):137–165, April 1993.