ARTICLE IN PRESS

ACADEMIC PRESS

Available at www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT

J. Parallel Distrib. Comput. ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Journal of Parallel and Distributed Computing

http://www.elsevier.com/locate/jpdc

Pipelined operator tree scheduling in heterogeneous environments

Arash Termehchi and Mohammad Ghodsi*

Parallel Processing Laboratory, Department of Computer Engineering, Sharif University of Technology, Tehran, Iran

Received 9 July 2001; revised 6 January 2003; accepted 19 February 2003

Abstract

Pipelined operator tree (POT) scheduling is an important problem in the area of parallel query optimization. A POT is a tree with nodes representing query operators that can run in parallel and edges representing communication between adjacent operators that is handled by sending long streams of data in a parallel-pipelined fashion. The problem is to find a schedule for the POT that minimizes the total response time. This problem has only been previously addressed for homogeneous environments, but the new parallel database systems tend to be more heterogeneous. In this paper, we consider processors with different fixed speeds (called uniform processor system). This problem has been shown to be NP-hard even for identical processors. We propose three approximate algorithms for some special cases of the problem with good low-performance ratio (or approximation factor) bound in the worst case. The performance ratios of these algorithms, even for the general case, are shown by experimentation to be near optimal on the average. We will show that the performance ratios of these algorithms, if used for homogeneous systems, are lower than the previous results. For the general case, we propose an algorithm which has a constant bound on the performance ratio in the worst case and is near optimal on the average.

1. Introduction

With emerging sophisticated applications on parallel database systems, such as decision support systems and data mining that access a large amount of data in most tables of the database, the need to minimize the query response time is more than ever. In such systems, the parallel query optimization which is to find the best execution plan of the queries, is more important and complicated than before. To reduce the complexity of this problem, some researchers have used a two-phase approach [8,13,16]: join ordering and query rewriting followed by parallelization and scheduling. In the first phase, the optimizations of the high-level query operations are performed, such as algebraic transformations, reordering of joins, etc. The annotated query tree generated in the first phase is then scheduled on the parallel machine, in the second phase. In this phase, atomic units of the query for parallel execution, called operators, are extracted first and then scheduled to provide the minimum response time. Many researchers have solved this problem only for special underlying hardware architectures and/or special groups of queries [5,16]. But, some others have considered this as a pure scheduling problem [10,11,14]. We use similar approach in this paper. One of the most important issues that must be considered is the parallelism–communication trade-off [6,7]. Scheduling two communicating operators on different processors can speed up query execution, but because of the involved communication cost, it can also increase the overall query execution time. To consider this trade-off, the query to be scheduled is represented as a weighted operator tree in which each node represents an operator and each edge represents the timing constraint between operators [14,16]. A timing constraint is either a precedence or parallel constraint.

The parallel constraint requires that the two adjacent nodes start and terminate their works approximately at the same time and behave as a producer–consumer system where the producer sends a long stream of communication data to the consumer.

A weighted operator tree in which all edges represent parallel constraints is called a pipelined operator tree (POT) [14]. POT scheduling problem is to find a schedule of the operators in POT that minimizes the

*Corresponding author.
E-mail addresses: termechi@ce.sharif.edu (A. Termehchi), ghodsi@sharif.ac.ir (M. Ghodsi).

0743-7315/03/$ - see front matter
doi:10.1016/S0743-7315(03)00034-0

total response time. The communication cost in the POT makes its scheduling different from the classical scheduling problems. The important part of the communication cost is the send and receive CPU overhead. This is because the data are transmitted in long streams. If adjacent nodes in POT are assigned to one processor, the communication cost between these nodes are saved, but this would decrease the degree of parallelism.

The POT scheduling problem was first introduced by Hasan and Motwani for identical processor systems and was shown to be NP-hard [14]. They proposed several approximation algorithms for restricted cases of the POT. Chekuri et al. devised two algorithms for the general case [4]. The first one, called LocalCuts, has the worst-case performance ratio of $\frac{3+\sqrt{17}}{2}$ and runs in $O(n \log n)$ time. The second, called BoundedCuts, has a smaller performance ratio of $(1+\varepsilon)2.87$, at the expense of higher time complexity of $O(\frac{1}{\varepsilon} n \log n)$. Chekuri has also shown that there exists a polynomial-time approximation schema (PTAS) for this problem [3]. This result is not practical because of its very high time complexity.

On the other hand, efficient approximate solution for parallel query optimization in heterogeneous processor systems has been considered to be a challenging problem [8,13]. Nowadays, there are many parallel database systems based on heterogeneous processors. Systems based on network of workstations or PC clusters are good examples of such systems [1,2,17,18]. The workstations may differ in processor speed, amount of memory and the speed and the number of attached disks. A practical example is given in [18] where a PC-cluster-based parallel database was used with heterogeneous processors. Moreover, the commodity interconnections such as Myrinet or ATM switches make pipelining a practical way and sometimes the only way to speed up query execution [2,17].

To the best of our knowledge, heterogeneity of resources has not been considered before in the context of the problem that we are dealing with. In this paper, we define POT scheduling on processors with different fixed speeds (called uniform processor system [12].). First, we extend LocalCuts and BoundedCuts algorithms for two uniform processors with lower worst-case performance ratio compared to the homogeneous case. We then extend LocalCuts heuristics for a more frequently occurring case of the problem (which is described later) with the worst-case performance ratio lower than the homogeneous case. We also propose an algorithm based on the LocalCuts heuristics to solve the problem in the general case and prove that the algorithm has a constant bound on its performance ratio. Experimental results show near-optimal average case performance ratios for all of our algorithms.

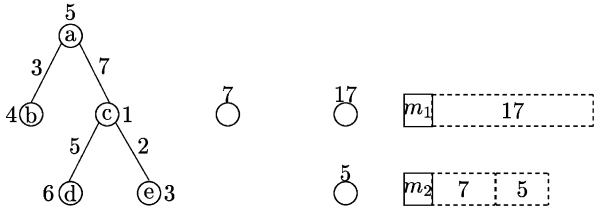
The organization of this paper is as follows. An overview of the model and problem definition are discussed in Section 2. In Section 3, we present our POT scheduling algorithms for a system with two uniform processors. The solutions for POT scheduling on an arbitrary number of uniform processors are presented in Section 4. Section 5 includes experimental results which follows with conclusions and future works.

2. The model and problem definition

The following definitions are based on earlier models presented in [4,14].

A POT is represented as a weighted operator tree $\mathscr{P} = (V, E)$. The weight p_k of the node k is the time to run the operator in isolation assuming all communications are local. The weight c_{kj} of the edge from node k to node j is the additional CPU overhead that both k and j will incur for interoperation communication if they are scheduled on different processors. A schedule of $\mathscr{P}$ on p processor is a partition of V, the set of nodes, into p sets $F_1, \ldots, F_p$ such that set F_k is assigned to processor k. The load of processor k, denoted by L_k, is the cost of executing all nodes in F_k plus the overhead for communicating with nodes on other processors. That is, $L_k = \sum_{j \in F_k} [p_j + \sum_{l \notin F_k} c_{jl}]$. $L_{\max}$ is $\max_{1 \leqslant k \leqslant p} L_k$.

In order to schedule the POT, two operations are used to modify it: *Collapse*(k,j) is to replace adjacent nodes k and j by a single node k' having weight of $t_{k'} = t_k + t_j$. Edges connected to either k or j are connected to k' instead. Operation *Cut*(k,j) is to delete edge e_{kj} and add its weight to those of node k and j. For example, in Fig. 2 the operations $Collapse(a,c)$, $Collapse(c,d)$, $Cut(a,b)$, and $Cut(c,e)$ have been performed. Collapse and cut operations should be interpreted as decisions to allocate nodes to the same or distinct processors, respectively. An edge e_{kj} is called *worthless* if and only if $c_{kj} \geqslant p_k + \sum_{l \neq j} c_{kl}$ or $c_{kj} \geqslant p_j + \sum_{l \neq k} c_{jl}$. As shown in [14] for a homogeneous case, each POT can be converted into a POT with no worthless edges, called monotone tree, by collapsing all its worthless edges. The monotone tree can then be scheduled. In a monotone tree, we use the following notations: $R_k = p_k + \sum_{j \in V} c_{kj}$, $R = \max R_k$, and $W = \sum_{k \in V} p_k$.



(a) The original POT (b) Fragmentation (c) Scheduling

Fig. 1. The two-stage approach: (a) the original POT; (b) fragmentation; (c) scheduling.

Our algorithms are based on a two-stage approach ([4], Fig. 1) which the following phases are performed: fragmentation, and the actual scheduling. This approach allows us to reuse classical multi-processor scheduling algorithms. In the fragmentation phase, the tree is partitioned into connected fragments by cutting some edges. The edges that are left are collapsed. Fragmentation produces a set of fragments that are ready to be scheduled independently. The scheduling phase assigns the fragments produced by the first phase. Let M_k be cost of a fragment F_k. Then, $M_k = \sum_{j \in F_k} [p_j + \sum_{l \notin F_k} c_{jl}]$. The weight of the heaviest fragment is denoted by M. Clearly, based on the definition of the monotone tree, R is a lower bound for M. C is the total communication cost incurred in a fragmentation which is twice the sum of the weights of the cut edges. So the total load L is $W + C$. In the algorithm presented, superscript $\star$ is used to denote the quantities in the optimal scheduling.

The problem that we solve is defined as follows. In a uniform processor system, the processors $m_1, \ldots, m_p$ have relative speeds of $s_1 \leqslant s_2 \leqslant \cdots \leqslant s_p$. We assume that these speeds have been normalized such that $s_1 = 1$ and $s_k \geqslant 1,\ 2 \leqslant k \leqslant p$. POT scheduling on a uniform processor system is to find a schedule with minimum response time of $T = \max_{1 \leqslant k \leqslant p} \frac{L_k}{s_k}$. Obviously, POT scheduling on uniform processors is also NP-hard. It is easily shown that we can collapse the worthless edges in the uniform processor case as well.

Theorem 2.1. *Given a uniform processor system and a* POT $\mathscr{P}$ *with worthless edge* e_{kj}. *There exists an optimal schedule of* $\mathscr{P}$ *for these processors in which k and j nodes are assigned to the same processor.*

3. Two uniform processor scheduling

We use the same two-stage approach. For scheduling, we apply LPT algorithm as has been used for the identical (homogeneous) case. LPT algorithm for the uniform processor system [12] assigns tasks to processors in the decreasing order of their processing times. A task is taken from this list and is assigned to a processor whose finishing time is the earliest. We first find a relationship between fragmentation phase and the LPT schedule. Then, we extend previous algorithms of the identical case.

3.1. Analysis of two-stage approach

As mentioned before, the fragmentation is done using collapse and cut operations. Clearly, heavy fragments increase load of processors and light fragments increase the communication cost of the edges cut. Therefore, a trade-off should be found for these effects.

Lemma 3.1. *Consider a fragmentation with* $M \leqslant k_1 L^\star_{\max}$ *and* $L \leqslant k_2 L^\star$ (k_1 *and* k_2 *are real values* $\geqslant 1$). *Scheduling the produced fragments by LPT on two uniform processors yields a schedule with* $T/T^\star \leqslant \max(k_1, 1.5k_2)$.

Proof. Let m_l denote the processor that finishes last, and M_k the lightest fragment assigned to m_l. Since LPT is used, M_k must be the last fragment assigned to m_l. We can remove all fragments which are lighter than M_k from the produced schedule without any change in T. Since $T^\star$ is fixed, there is no change in the performance ratio. Clearly, L', the total load of the new set of fragments, is less than or equal to L, and the weight of the largest fragment M remains unchanged. For different values of k, the following cases occur:

1. If $k \geqslant 2$, from the definition of LPT, and scheduling M_k, we have

$$\forall 1 \leqslant j \leqslant 2 : T \leqslant \frac{L'_j + M_k}{s_j} \Rightarrow T\left(\sum_{j=1}^{2} s_j\right) \leqslant \sum_{j=1}^{2} L'_j + 2M_k,$$

in which L'_j is the load of m_j, when M_k is scheduled. Since $\sum_{j=1}^{2} L'_j = L' - M_k$ and $L' \leqslant L \leqslant k_2 L^\star$, we have, $T \leqslant k_2 L^\star / \sum_{j=1}^{2} s_j + M_k / \sum_{j=1}^{2} s_j$.

Because $L^\star / \sum_{j=1}^{2} s_j \leqslant T^\star$, we get $T \leqslant k_2 T^\star + \frac{M_k}{L^\star} T^\star$. And thus $T \leqslant k_2 T^\star + \frac{k_2 M_k}{L'} T^\star$. Since M_k is the lightest task in the schedule, we have $L' \geqslant k M_k$. Therefore, $T \leqslant k_2 T^\star + \frac{k_2}{k} T^\star$. Thus, $T \leqslant 1.5 k_2 T^\star \leqslant \max(k_1, 1.5k_2) T^\star$.

2. If $k = 1$, M_k will be assigned to processor with maximum speed. Therefore, $T \leqslant \frac{M_k}{s_2}$. From assumption of lemma, we conclude that $T \leqslant k_1 L^\star_{\max}/s_2 \leqslant k_1 T^\star \leqslant \max(k_1, 1.5k_2) T^\star$. □

Algorithms are thus needed for fragmentation of the POT with minimum values of k_1 and k_2.

3.2. Modification of previous algorithms

We modify LocalCuts algorithm based on our analysis for two uniform processor case. LocalCuts repeatedly picks up a leaf and determines whether to cut or collapse the edge from the leaf to its parent. It selects a proper operation based on ratio of the leaf weight to the weight of the edge to its parent. If the ratio is greater than an input parameter $\alpha > 1$, it will cut the edge, since this operation does not considerably increase the weight of the resulting fragments. If the ratio is less than α, the leaf is collapsed to the parent node. This is because the

weight of the parent node will not increase substantially. In the algorithm, a *mother node* is defined as a node all of whose children are leaves. In the algorithm that follows, we choose a proper value for α based on Theorem 3.1.

ALGORITHM 1 (LOCALCUTS ALGORITHM).

while there exist a mother node m with child j
  if $p_j > \alpha c_{jm}$
    then Cut(j,m)
    else Collapse(j,m)
end while

Theorem 3.1. *The worst-case performance ratio of* LOCALCUTS *followed by* LPT *for scheduling on two uniform processors is* 3.

Proof. We have $C \leqslant 2/(\alpha-1)W$ in LOCALCUTS algorithm [4]. Therefore, $L = W + C \leqslant \frac{\alpha+1}{\alpha-1} W$. Since $W \leqslant L^\star$, then $L \leqslant \frac{\alpha+1}{\alpha-1} L^\star$. We also have $M < \alpha R$ for LOCALCUTS fragmentation [4]. Using Lemma 3.1, we get $\frac{T}{T^\star} \leqslant \max(\alpha, 1.5\frac{\alpha+1}{\alpha-1})$.

Because $1.5\frac{\alpha+1}{\alpha-1}$ is strictly decreasing in α and α is itself strictly increasing in α, minimizing the right-hand side of the above inequality gives us $\alpha = 1.5\frac{\alpha+1}{\alpha-1}$, which leads to $\alpha = 3$. Thus, the worst-case performance ratio of the algorithm is also equal to 3. □

We can show that this performance ratio is tight with scheduling a POT similar to the example in [4] on two processors with equal speeds.

The second algorithm that we modify is BOUNDEDCUTS. If R is small compared to $M^\star$, LOCALCUTS may cut expensive edges needlessly (maximum weight of fragments produced by LOCALCUTS is bounded by αR). This was the reason Chekuri, et al., modified LOCALCUTS using a uniform bound B at each mother node. The new algorithm, BOUNDEDCUTS, fragments POT based on three parameters α, β, and B that satisfy $\beta \geqslant \alpha > 1$ and $L^\star \leqslant B \leqslant (1+\varepsilon)L^\star \varepsilon > 1$. This algorithm cuts off light edges in a manner similar to LOCALCUTS. But it collapses edges based on αB bound. The reader is referred to [3] on how the value of B is chosen.

ALGORITHM 2 (BOUNDEDCUTS ALGORITHM).

while there exist a mother node m
  Partition children of m into sets N_1, N_2 such that
    child $j \in N_1$ iff $\frac{p_j}{c_{mj}} \geqslant \beta$
  Cut(m,j) for all $j \in N_1$
    then Collapse(m,j) for all $j \in N_2$
    else Cut(m,j) for all $j \in N_2$
end while

We now extend BOUNDEDCUTS algorithm choosing the proper value for α and β.

Theorem 3.2. *The worst-case performance ratio of* BOUNDEDCUTS *followed by* LPT *algorithm for scheduling on two uniform processors* $2.35(1+\varepsilon)$.

Proof. From [4] we know, $C \leqslant \frac{2}{\beta-1} W + \frac{\beta-\alpha}{\alpha-1} C^\star$ for BOUNDEDCUTS algorithm. Since $L = W + C$, we have, $L \leqslant \max(\frac{\beta+1}{\beta-1}, \frac{\beta-\alpha}{\alpha-1}) L^\star$.

From [4], we know that $M \leqslant (1+\varepsilon)\alpha L^\star_{\max}$. Using Lemma 3.1, we get, $\frac{T}{T^\star} \leqslant \max((1+\varepsilon)\alpha, 1.5\frac{\beta+1}{\beta-1}, 1.5\frac{\beta-\alpha}{\alpha-1})$. Because $1.5\frac{\beta+1}{\beta-1}$ is strictly decreasing in β, $1.5\frac{\beta-\alpha}{\alpha-1}$ is strictly increasing in β and decreasing in α, and $(1+\varepsilon)\alpha$ is strictly increasing in α, in order to minimize the right-hand side of the above inequality, the values of these functions must be equal. We thus have $\alpha = 2.35$ and $\beta = 4.51$. Therefore, the performance ratio is $(1+\varepsilon)2.35$. □

Similar to [4], we can prove the performance ratio of the algorithm is tight.

4. P-uniform processor scheduling

In this section, we first extend LOCALCUTS heuristic followed by LPT algorithm for a frequent case of the problem. We then prove that the combination of LOCALCUTS followed by any scheduling algorithm with a constant bound on its performance ratio (i.e., LPT or Multifit [9]), provides a performance ratio with a constant bound.

4.1. The frequent case

If W is distributed approximately uniformly among nodes of the POT, maximum degree is considerably less than the number of nodes n, and also $p < n$, we can prove that LOCALCUTS followed by LPT has the worst-case performance ratio in the interval of $[3 \ldots \frac{3+\sqrt{17}}{2})$. These conditions often occur in scheduling parallel database queries. In such cases, POT is usually a binary or at most 3-ary tree [13] and the costs of pipelined operations do not differ very much. This is because the heavy operators are usually partitioned among some processors (partitioned parallelism). Many of the parallel database applications such as data mining systems often process sophisticated queries whose POTs have high number of nodes. These conditions on a POT can be depicted mathematically as $W \geqslant 2(p-1)R$.

Lemma 4.1. *Assume a uniform processor system with p processor and a* POT $\mathscr{P}$ *in which* $L^\star \geqslant 2(p-1)R$. *If a fragmentation algorithm fragments* $\mathscr{P}$ *such that* $M \leqslant k_1 R$

and $L \leqslant k_2 L^\star$, applying LPT *on the produced fragments gives a performance ratio of* $\max(k_1, (2-\frac{1}{p})k_2)$.

Proof. Similar to proof of Lemma 3.1, we first eliminate all fragments lighter than M_k from the schedule produced by LPT. As explained before, this does not change the performance ratio of the schedule and M, and also may decrease the total load L'. Based on the definition of LPT in time of scheduling M_k, we have,

$$\forall 1 \leqslant j \leqslant p : T \leqslant \frac{L'_j + M_k}{s_j}.$$

L'_j is the load of processor m_j when M_k is scheduled. Similar to case 1 in proof of Lemma 3.1, and summing up the above p inequalities, we conclude that

$$T \leqslant \frac{L'}{\sum_{j=1}^{p} s_j} + \frac{(p-1)M_k}{\sum_{j=1}^{p} s_j}.$$

Based on the values of k, we have the following cases:

1. If $k \geqslant p$, similar to proof of Lemma 3.1, it can be proved that $T \leqslant k_2 T^\star + \frac{(p-1)k_2}{k} T^\star$. Since $k \geqslant p$, we have $T \leqslant k_2 T^\star + \frac{(p-1)k_2}{p} T^\star$. Then, $T \leqslant k_2(2-\frac{1}{p})T^\star$.
2. If $k < p$, we have $L' = \sum_{l=1}^{k} M_l \leqslant \sum_{l=1}^{p-1} M_l$. Since M is the heaviest fragment, we have $L' \leqslant (p-1)M$. Thus, $T \leqslant \frac{(p-1)M}{\sum_{j=1}^{p} s_j} + \frac{(p-1)M_k}{\sum_{j=1}^{p} s_j}$. And because $M_k \leqslant M$, we conclude that $T \leqslant 2(p-1)M / \sum_{j=1}^{p} s_j$. From the assumption of lemma, we have $T \leqslant 2(p-1)k_1 R / \sum_{j=1}^{p} s_j$. From $L^\star \geqslant 2(p-1)R$, we conclude that $T \leqslant \frac{k_1 L^\star}{\sum_{j=1}^{p} s_j}$, which leads to $T \leqslant k_1 T^\star$.

Therefore, for the general case we have, $T \leqslant \max(k_1, (2-\frac{1}{p})k_2)T^\star$. □

Now we can prove the following theorem regarding the performance ratio of LocalCuts followed by LPT.

Theorem 4.1. *The worst-case performance ratio of* LocalCuts *followed by LPT for scheduling the* POT $\mathscr{P}$, *in which $W \geqslant 2(p-1)R$, on a uniform processor system lies in the interval of* $[3 \cdots \frac{3+\sqrt{17}}{2})$, *depending on the number of processors.*

Proof. Because of $W \leqslant L^\star$, we can use Lemma 4.1. Similar to the proof of Theorem 3.1, we can prove that $\frac{T}{T^\star} \leqslant \frac{1}{2}(3 - \frac{1}{p} + \sqrt{17 + \frac{1}{p^2} - \frac{10}{p}})$, by selecting $\alpha = \frac{1}{2}(3 - \frac{1}{p} + \sqrt{17 + \frac{1}{p^2} - \frac{10}{p}})$. This performance ratio always lies in the interval $[3 \cdots \frac{3+\sqrt{17}}{2})$. □

We can calculate the value of W and R in linear time to test conditions of the input POT and apply the above algorithm. Similar to the previous case, we can show that this performance ratio is tight.

4.2. The general case

To solve the problem in the general case, a mapping between the optimal and an arbitrary schedule is used.

Lemma 4.2. *Suppose that there exists a mapping function f from fragments produced by a fragmentation π of the POT $\mathscr{P}$ to fragments of the optimal schedule, such that it satisfies the following conditions*:

1. *f is total on its domain, and*
2. *for all F_k $(k \geqslant 1)$, such that $f(F_k) = F_j^\star$ $(j \geqslant 1)$, we have $\frac{\sum M_k}{M_j^\star} \leqslant r$ $(r > 0)$.*

Then, scheduling the fragments of π using an uniform processor scheduling algorithm, A_s, with the worst-case performance ratio of ε has performance ratio less than or equal to $r\varepsilon$.

Proof. First, we make a schedule S_1 from the optimal schedule π. Each F_k fragment of π is assigned to the processor that $f(F_k)$ has been assigned to in the optimal schedule. Because of totality of f, we can schedule all fragments of π in this manner. From our assumption, if T_1 is the response time of S_1, we have $\frac{T_1}{T^\star} \leqslant r$. Clearly, if $T_1^\star$ is the response time of the optimal schedule of fragments produced by π, we have $T_1^\star \leqslant T_1$. Therefore, scheduling these fragments by A_s yields a response time of $T \leqslant \varepsilon T_1^\star \leqslant \varepsilon T_1$. Then $\frac{T}{T^\star} \leqslant r\varepsilon$. □

So an algorithm is needed for fragmentation such that we can define a function with the above condition with constant and minimum value of r. The following theorem proves that LocalCuts heuristics satisfy the above conditions and provides a constant value for r. We first need the following definition.

Definition 4.1. The *main node* of a fragment F is a node whose parent does not belong to F. The main node of F is denoted by $\mu(F)$.

In other words, the main node of a fragment is the highest level node in that fragment. Clearly, every fragment has one and only one main node.

Theorem 4.2. *The worst-case performance ratio of* LocalCuts *followed by the uniform processor scheduling algorithm that has the worst-case performance ratio of ε is 8ε.*

Proof. We define relation f from fragments of LocalCuts to those of optimal solution such that $f(F_k) = F_j^\star$, if and only if $\mu(F_k)$ belongs to F_j. Clearly, f is a total function on its domain because each fragment of LocalCuts has one and only one main node and each node of the POT belongs to one and only one fragment in its optimal solution.

The value of $r = \frac{\sum M_k}{M_j^\star}$, such that $\forall k\ f(F_k) = \bar{F}_j^\star$, is maximized when $\sum M_k$ is maximized and $M_j^\star$ is

minimized. Assume that f maps n fragments $F_1, \ldots, F_n$ to one fragment $F_j^\star$. Because of the definition of f, $F_j^\star$ must have at least n nodes. $F_1, \ldots, F_n$ must be connected because they are mapped into a fragment. $M_j^\star$ is minimized, if $F_j^\star$ cuts all incident edges of the main nodes of $F_1, \ldots, F_n$, except those edges that connect the main nodes to each other (see Fig. 2). This is because the POT is monotone and for every edge e_{kj} we have, $c_{kj} < R_k - c_{kj}$. $M_j^\star$ is also minimized if $F_j^\star$ collapses all edges which connect $\mu(F_1), \ldots, \mu(F_n)$ to each other as is shown in the figure. We therefore have, $\sum_{k=1}^{n} M_k = \sum_{k=1}^{n} [m_k + \sum_{j=1}^{q} p_j + \sum_{l=1}^{u} c_l]$, in which m_k is the weight of $\mu(F_k)$, p_j is the weight of the child node j of $\mu(F_k)$ that belongs to F_k, and c_l is the weight of the connecting edge between fragments incident $\mu(F_k)$. For each boundary node j of F_k, we assume that m_j and p_j are the sum of node weights plus the weights of all edges incident to j that are not in F_k.

Because $F_1, \ldots, F_n$ form a subtree, and each connecting edge between F_k and F_j is considered for computing the cost of both F_k and F_j, we have, $\sum_{k=1}^{n} M_k = \sum_{k=1}^{n} [m_k + \sum_{j=1}^{q} p_j] + 2\sum_{l=1}^{n-1} c_l$.

Since connecting edges are cut by LocalCuts, we have

$$\sum_{k=1}^{n} M_k < \sum_{k=1}^{n} \left[m_k + \sum_{j=1}^{q} p_j \right] + \frac{2}{\alpha} \sum_{k=2}^{n} M_k.$$

Let $\alpha > 2$ so

$$\sum_{k=1}^{n} M_k < \frac{\alpha}{\alpha - 2} \sum_{k=1}^{n} \left[m_k + \sum_{j=1}^{q} p_j \right].$$

For $M_j^\star$ we have, $M_j^\star = \sum_{k=1}^{n} [m_k + \sum_{j=1}^{q} c_j]$, in which c_j is the weight of the cut edge j incident to $\mu(F_k)$. Note that j cannot be a connecting edge between the main nodes. Since these edges are collapsed by LocalCuts, we have $M_j^\star \geqslant \sum_{k=1}^{n} [m_k + \frac{1}{\alpha}\sum_{j=1}^{q} p_j]$.

To compute the ratio, we have

$$\frac{\sum_{k=1}^{n} M_k}{M_j^\star} < \frac{\sum_{k=1}^{n} [m_k + \sum_{j=1}^{q} p_j]}{\sum_{k=1}^{n} [m_k + \frac{1}{\alpha} \sum_{j=1}^{q} p_j]} \left(\frac{\alpha}{\alpha - 2} \right) < \frac{\alpha^2}{\alpha - 2}.$$

The above ratio is minimized when $\alpha = 4$, and the minimum value of r is 8. This completes the proof. □

For the scheduling phase, Multifit algorithm can be used, since its worst-case performance ratio is 1.3 [9].

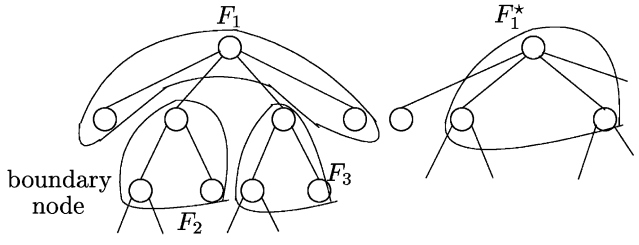


Fig. 2. The mapping between LocalCuts and optimal fragmentation.

Corollary 4.1. *The worst-case performance ratio of* LocalCuts *followed by* Multifit *algorithm for scheduling on a uniform processor system is* 10.4.

We can also use the PTAS of Hochbaum and Shmoys [15].

Corollary 4.2. *The worst-case performance ratio of* LocalCuts *followed by the PTAS of the uniform processors scheduling of Hochbaum and Shmoys is* $8(1+\varepsilon)$.

The first algorithm is more practical compared to the second one. Because Multifit takes $O(n \log n)$, PTAS has a high running time of $O(n^{\frac{10}{2}+3})$.

5. Experimental results

Experimental simulation has been used to analyze the average case performance ratios of our proposed algorithms. We basically used the same model as in [13] with some modifications.

The trees that are generated randomly for our simulation are specified by four parameters: Shape, Size, EdgeRange, and NodeRange. Shape is the maximum number of children of a node in the tree. Two main classes of tree shapes are called *narrow* and *wide*. Narrow trees are binary trees, but wide trees can have any number of children for a node. Narrow trees are commonly encountered in practice. EdgeRange and NodeRange are ranges from which the weights for edges and nodes could be chosen. Size is the number of nodes in a tree. From the trees randomly generated, we only selected those that are monotone.

Each processor system is specified by two parameters: SpeedRange and SpeedDistribution. SpeedRange is the difference between the slowest and the fastest processors. All other speeds are chosen in this range. SpeedDistribution is the distribution of speeds of processors.

We chose 30 for Size parameter and [1…100] for both EdgeRange and NodeRange. Other values did not yield new results. Reported performance ratios are the average values for 2500 monotone trees generated with these parameters. We also chose a small value (3) for SpeedRange and uniform distribution for SpeedDistribution; these values are common in practice. We tested our algorithms with different values of SpeedRange and SpeedDistribution. This slightly changed the average case performance ratios, but did not change the relative order of the performance of the algorithms.

Computating the optimal schedule is very time consuming. We therefore used the maximum values of the two lower bounds of the optimal solutions for the

optimal values. These two bounds are $\frac{W}{\sum s}$ and $\frac{R}{\max s}$ as described in the paper.

Figs. 3 and 5 depict the performance ratios of the BOUNDEDCUT algorithm for narrow and wide trees, respectively, and for a number of processors from 2 to 30. Similar plots for LOCALCUTS algorithms are shown in Figs. 4 and 6. The experimental result for this algorithm has been computed for three different values of α $(3, \frac{1}{2}(3-\frac{1}{p}+\sqrt{17+\frac{1}{p^2}-\frac{10}{p}})$, and 4) which are also shown. For the first two values of α, we have used LPT scheduling algorithm, and we have used MULTIFIT for the last.

The analysis of the results is as follows. We know that the first lower bound mentioned before is close to the optimal response time when the number of nodes is considerably greater than $\sum s$. This can also happen when the number of nodes is greater than the number of processors for a specified SpeedRange. This case is shown in the left regions of the experimental plots. The other lower bound is also close to the optimal response time when the number of nodes is less than or approximately equal to the number of processors for a specified SpeedRange. The right regions of the plots demonstrate this case. The middle regions of the plots show that the mentioned lower bounds are not good estimates for the optimal solution. Therefore, our plots show some increase in the performance ratios in the middle region. We can conclude that, for wide trees, $\frac{R}{\max s}$ is a better estimate for the optimal response time than that for the narrow trees. This is because R is closer to $M^{\star}$ in wide trees. This results in the fact that the middle regions of the plots for the wide trees are narrower than those for the narrow trees.

From the first two plots, we can conclude that the average performance ratios of the BOUNDEDCUTS algorithm are very close to the optimal values and are in many cases smaller than other three LOCALCUTS algorithms. It is also observed that this algorithm's performance in the right region is much better than those of other algorithms. The reason is that the algorithm generates lighter-weight fragments in this case compared to others.

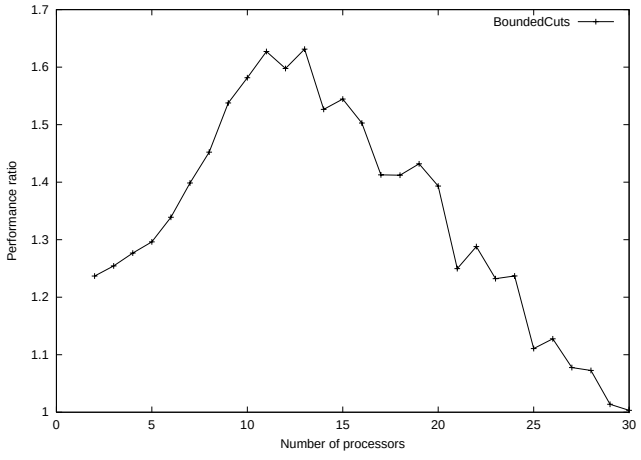


Fig. 3. Average performance ratios of BOUNDEDCUTS for narrow trees.

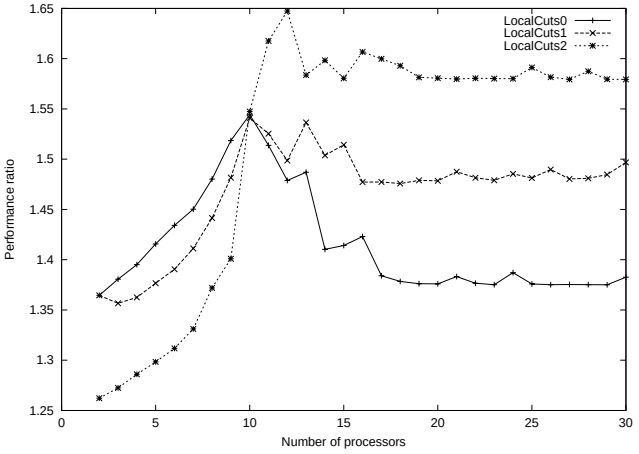


Fig. 4. Average performance ratios of LOCALCUTS for narrow trees.

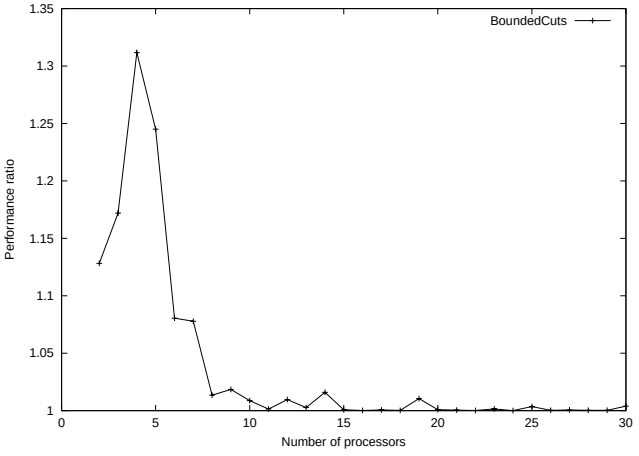


Fig. 5. Average performance ratios of BOUNDEDCUTS for wide trees.

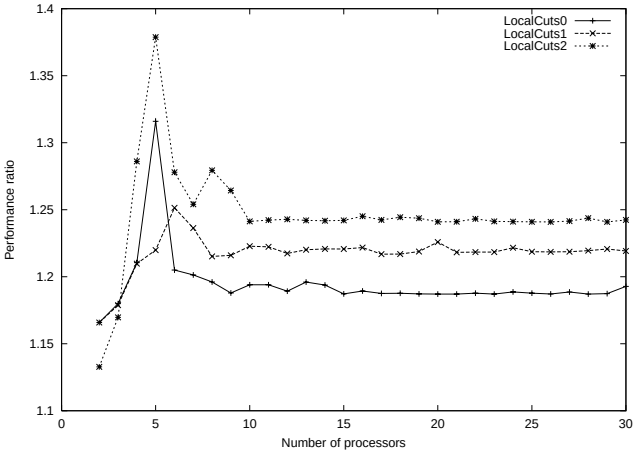


Fig. 6. Average performance ratios of LOCALCUTS for wide trees.

We have also tested BOUNDEDCUTS for some other values of ε in the interval of $[0.1 \ldots 0.01]$, but no considerable changes in the performance ratios were observed.

From these experimentations, we conjecture that the worst-case performance ratio of BOUNDEDCUTS is about 3 for the general case.

For LOCALCUTS, we formally proved that its performance ratio is at most $\varepsilon \frac{\alpha^2}{(\alpha-2)}$, $\varepsilon \geqslant 1$. We experimentally observe that the actual average case performance ratios of all versions of LOCALCUTS are close to the optimal values.

Other observation is that for the small values of α, LOCALCUTS generates lighter-weight fragments which results in smaller performance ratios in the right region and larger performance ratios in the left region. This helps us select the appropriate values for α in different regions. The borderlines between the regions is determined by experimentation. We believe that this will improve the algorithm even further.

6. Conclusions

In this paper, we studied the heterogeneity of resources in parallel query scheduling. This is an important concern and has not been considered in previous similar works. We introduced the POT scheduling on the uniform processor system and proposed approximation algorithms for some special cases of the problem as well as for the general case. We showed that our algorithms have constant bounds on their worst-case performance ratio. The performance ratios of these algorithms, even for the general case, are shown by experimentation to be near optimal on the average.

This problem can be extended in many ways any other properties of the parallel database systems can be taken into account for these problems. On heterogeneous network of workstations, for example, heterogeneity of other resources and consideration of the propagation delay time between workstations are important. Some preliminary results can be found in [19]. Further work on this is underway.

References

[1] R. Bamford, D. Butler, B. Klots, N. Macnaughton, Architecture of Oracle parallel server, in: Proceedings of the 24th International Conference on Very Large Data Bases, New York, USA, August 1998, pp. 669–670.

[2] L. Brunie, M. Exbrayat, A. Flory, A coupled approach for parallel evaluation of relational queries on a network of workstations, XIV. International Symposium on Computer and Information Sciences, Kusadas, Turkey, October 1999.

[3] C. Chekuri, Approximation algorithms for scheduling problems, Ph.D. Thesis, Department of Computer Science, Stanford University, 1998.

[4] C. Chekuri, W. Hasan, R. Motwani, Scheduling problems in parallel query optimization, in: Proceedings of the 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principle of Database Systems, San Jose, CA, May 1995, pp. 255–265.

[5] M. Chen, M. Lo, P.S. Yu, H.C. Young, Applying segmented right-deep trees to pipelining multiple hash joins, IEEE Trans. Knowledge Data Eng. 7 (4) (1995) 656–668.

[6] D.J. Dewitt, J. Gray, Parallel database system: the future of high performance database systems, Comm. ACM 35 (6) (1992) 85–98.

[7] S. Englert, R. Glasstone, W. Hasan, Parallelism and its price: a case study of nonstop SQL/MP, ACM SIGMOD Record, December 1995.

[8] D. Florescu, W. Hasan, P. Valdurize, Open issues in parallel query optimization, ACM SIGMOD Record, Vol. 25(3), September 1996, pp. 28–33.

[9] D. Friesen, M. Langston, Bounds for MULTIFIT scheduling on uniform processors, SIAM J. Comput. 12 (1) (1983) 60–70.

[10] M.N. Garofalaksis, Y.E. Ioannidis, Multi-dimensional resource scheduling for parallel queries, in: Proceedings of ACM SIGMOD International Conference on Management of Data, June 1996, pp. 365–376.

[11] M.N. Garofalakis, Y.E. Ioannidis, Parallel query scheduling and optimization with time and space-shared resources, in: Proceedings of the 23rd International Conference on Very Large Data Bases, Athene, Greece, August 1997, pp. 296–305.

[12] T. Gonzalez, O. Ibarra, S. Sahni, Bounds for LPT schedule on uniform processors, SIAM J. Comput. 6 (1) (1977) 150–164.

[13] W. Hasan, Optimization of SQL queries for parallel machines, Ph.D. Thesis, Department of Computer Science, Stanford University, December 1996.

[14] W. Hasan, R. Motwani, Optimization algorithms for exploiting the parallelism, communication tradeoff in pipelined parallelism, in: Proceedings of the 20th International Conference on Very Large Data Bases, Santiago, Chile, September 1994, pp. 36–47.

[15] D. Hochbuam, D. Shmoys, A polynomial approximation scheme for scheduling on uniform processors: using the dual approximation approach, SIAM J. Comput. 17 (3) (1988) 539–553.

[16] W. Hong, Parallel query processing using shared memory multiprocessors and disk arrays, Ph.D. Thesis, Department of Computer Science, University of California, Berkeley, August 1992.

[17] M. Kitsuregawa, M. Oguchi, T. Tamura, Parallel database processing on a 100 node PC cluster: cases for decision support query processing and data mining, in: Proceedings of the International Conference for High Performance Computing and Communications, San Jose, CA, November 1997.

[18] M. Kitsuregawa, T. Tamura, Dynamic load balancing for parallel association rule mining on heterogeneous PC cluster system, in: Proceedings of the 25th International Conference on Very Large Data Bases, Edinburgh, Scotland, September 1999, pp. 162–173.

[19] A. Termehchi, Query optimization for parallel execution, MS Thesis, Department of Computer Engineering, Sharif University of Technology, Tehran, October 1999.