Multi-dimensional Resource Scheduling for Parallel Queries

Minos N. Garofalakis
Computer Sciences Department
University of Wisconsin
Madison, WI 53706
minos@cs.wisc.edu

Yannis E. Ioannidis*
Computer Sciences Department
University of Wisconsin
Madison, WI 53706
yannis@cs.wisc.edu

Abstract

Scheduling query execution plans is an important component of query optimization in parallel database systems. The problem is particularly complex in a shared-nothing execution environment, where each system node represents a collection of time-shareable resources (e.g., CPU(s), disk(s), etc.) and communicates with other nodes only by message-passing. Significant research effort has concentrated on only a subset of the various forms of intra-query parallelism so that scheduling and synchronization is simplified. In addition, most previous work has focused its attention on one-dimensional models of parallel query scheduling, effectively ignoring the potential benefits of resource sharing. In this paper, we develop an approach that is more general in both directions, capturing all forms of intra-query parallelism and exploiting sharing of multi-dimensional resource nodes among concurrent plan operators. This allows scheduling a set of independent query tasks (i.e., operator pipelines) to be seen as an instance of the multi-dimensional bin-design problem. Using a novel quantification of coarse grain parallelism, we present a list scheduling heuristic algorithm that is provably near-optimal in the class of coarse grain parallel executions (with a worst-case performance ratio that depends on the number of resources per node and the granularity parameter). We then extend this algorithm to handle the operator precedence constraints in a bushy query plan by splitting the execution of the plan into synchronized phases. Preliminary performance results confirm the effectiveness of our scheduling algorithm compared both to previous approaches and the optimal solution. Finally, we present a technique that allows us to relax the coarse granularity restriction and obtain a list scheduling method that is provably near-optimal in the space of all possible parallel schedules.

1 Introduction

Parallelism has been recognized as a powerful and cost-effective means of handling the projected increases in data size and query complexity in future database applications. Among all proposals, the shared-nothing multiprocessor architecture has emerged as the most scalable to support very large database management [DG92]. In this, each *site* consists of its own set of local resources and communicates with other sites only by message-passing. Despite the popularity of this architecture, the development of effective and efficient query processing and optimization techniques to exploit its full potential still remains an issue of concern.

Earlier work on parallel query scheduling has typically concentrated on two important problems:

1. *compile-time optimization:* minimizing the response time of a single query through *parallelization* of an execution plan, i.e., scheduling of the plan's operators on the system's sites (the plan is usually the result of an earlier phase of conventional centralized query optimization) [CHM95, GW93, HM94, Hon92, HCY94, LCRY93]; and
2. *run-time execution:* achieving some system-wide performance goals (e.g., maximizing query throughput) by adaptive scheduling of the operators of multiple concurrent queries [MD93, MD95, RM95].

We address the first problem, i.e., parallelization of query execution plans. We consider the full variety of bushy plans and schedules that incorporate *independent* and *pipelined* forms of inter-operation parallelism as well as intra-operation (i.e., *partitioned*) parallelism.

One of the main sources of complexity of query plan scheduling is the *multi-dimensionality* of the resource needs of database queries. That is, during their execution queries alternate between multiple resources, most of which are *preemptable* [GHK92], e.g., the CPU and disk bandwidth. This introduces a range of possibilities for effectively time-sharing system resources among concurrent query operators, which can substantially increase the utilization of these resources and reduce the response time of the query.

Previous work on parallel query scheduling has typically ignored the multi-dimensional nature of database queries. It has simplified the allocation of resources to a mere allocation of processors, hiding the multi-dimensionality of query operators under a scalar cost metric like "work" or "time" [CHM95, GW93, HM94, HCY94, LCRY93]. This one-dimensional model of scheduling is inadequate for database operations that impose a significant load on multiple system resources.

In this paper, we present a framework for multi-dimensional resource scheduling in shared-nothing parallel

*Partially supported by the National Science Foundation under Grants IRI-9113736 and IRI-9157368 (PYI Award), and by grants from IBM, DEC, HP, AT&T, Informix, and Oracle.

SIGMOD '96 6/96 Montreal, Canada
© 1996 ACM 0-89791-794-4/96/0006...$3 50

database systems. Building on the work of Ganguly et al. [GHK92], we represent query operator costs as *work vectors* with one dimension per resource. In order to account for the communication overhead of parallelism, we initially restrict our attention to operator parallelizations that are sufficiently coarse grain. We present a quantification of the notion of coarse granularity based on the relative costs of communication and computation and use it to derive the degree of partitioned parallelism.

Based on this framework, the problem of resource scheduling for a collection of concurrently executed operators is reduced to an instance of the *multi-dimensional bin-design problem* [CGJ84] for work vector packings. For this, we develop a fast resource scheduling algorithm called OPERATORSCHEDULE that belongs to the class of *list scheduling algorithms* [Gra66]. The *response time* (or, *makespan*) of the parallel schedule produced by OPERATORSCHEDULE is analytically shown to be

(a) within $(2d+1)$ of the optimal schedule length for given degrees of partitioned parallelism, and

(b) within $(2d(fd+1)+1)$ of the optimal coarse grain schedule length,

where d is the dimensionality of the work vectors and f is a "small" parameter capturing the granularity of the parallel execution. We then extend the algorithm to handle the operator precedence constraints in a bushy query plan by splitting the execution of the plan into synchronized phases. The resulting algorithm, called TREESCHEDULE, uses OPERATORSCHEDULE as a subroutine to determine the scheduling of operators within each phase. Preliminary experimental results confirm the effectiveness of these algorithms compared to previous one-dimensional approaches. In addition, our results show that the analytical worst-case bounds are rather pessimistic compared to the average performance, which is extremely close to optimal. Finally, we consider the more general *malleable* problem in which the solution is no longer constrained by a coarse granularity condition. Instead, the scheduler is free to determine the degrees of partitioned parallelism with the objective of minimizing response time over *all possible* parallel schedules. Building on the ideas of Turek et al. [TWY92] we present a technique that allows our list scheduling rule for independent operators to achieve a suboptimality bound of $(2d+1)$ for the malleable problem at the additional cost of a preprocessing parallelization step.

2 Related Work

The problem of scheduling complex query plans on parallel machines has recently attracted a lot of attention from the database research community. Hasan and Motwani [HM94] study the tradeoff between pipelined parallelism and its communication overhead and develop near-optimal heuristics for scheduling a star or a path of pipelined relational operators on a multiprocessor architecture. Chekuri et al. [CHM95] extend these results to arbitrary pipelined operator trees. The heuristics proposed in these papers ignore both independent and partitioned parallelism. Ganguly and Wang [GW93] describe the design of a parallelizing scheduler for a tree of coarse grain operators. Based on a *one-dimensional* model of query operator costs, the authors show their scheduler to be near-optimal for a limited space of query plans (i.e., left-deep join trees with a single materialization point in any right subtree). Ganguly et al. [GGW95] obtain similar results for the problem of partitioning independent pipelines without the coarse granularity restriction. The benefits of resource sharing and the multi-dimensionality of query operators are not addressed in these papers. Furthermore, no experimental results are reported. Lo et al. [LCRY93] develop optimal schemes for assigning processors to the stages of a pipeline of hash-joins in a shared-disk environment. Their schemes are based on a *two-phase minimax* formulation of the problem that ignores communication costs and prevents processor sharing among stages. Moreover, no methods are proposed for handling multiple join pipelines (i.e., independent parallelism).

With the exception of the papers mentioned above, most efforts are experimental in nature and offer no theoretical justification for the algorithms that they propose. In addition, many proposals have simplified the scheduling issues by ignoring independent (bushy tree) parallelism; these include the right-deep trees of Schneider [Sch90] and the segmented right-deep trees of Chen et al. [CLYY92]. Nevertheless, the advantages offered by such parallelism, especially for large queries, have been demonstrated in prior research [CYW92].

Tan and Lu [TL93] and Niccum et al. [NSHL95] consider the general problem of scheduling bushy join plans on parallel machines exploiting all forms of intra-query parallelism and suggest heuristic methods of splitting the bushy plan into non-overlapping *shelves* of concurrent joins. For the same problem, Hsiao et al. [HCY94] propose a processor allocation scheme based on the concept of *synchronous execution time*: the set of processors allotted to a parent join pipeline are recursively partitioned among its subtrees in such a way that those subtrees can be completed at approximately the same time. For deep execution plans, there exists a point beyond which further partitioning is detrimental or even impossible, and serialization must be employed for better performance. Wilschut et al. [WFA95] present a comparative performance evaluation of various multi-join execution strategies on the PRISMA/DB parallel main-memory database system.

A common characteristic of all approaches described above is that they consider a one-dimensional model of resource allocation based on a scalar cost metric (e.g., "work"), which ignores any possibilities for effective resource sharing among concurrent operations. Perhaps the only exception is Hong's method for exploiting independent parallelism in the XPRS shared-memory database system [Hon92]. He suggests a scheduling algorithm that combines one I/O-bound and one CPU-bound operator pipeline through independent parallelism to maximize the system resource utilizations and thus minimize the elapsed time. Hong's algorithm depends on the dynamic (run-time) adjustment of intra-operator parallelism to ensure that the system always executes at its IO-CPU balance point. However, this approach may fail in the context of a shared-nothing architecture since the substantial communication overhead involved in relation declustering causes the cost of dynamic load balancing to increase dramatically.

Moving away from the database field, there is a significant body of work on parallel task scheduling in the field of deterministic scheduling theory. Since the problem is $\mathcal{NP}$-hard in the strong sense [DL89], research efforts have concentrated on providing fast heuristics with provable worst case bounds on the suboptimality of the solution. However, scheduling query plans on shared-nothing architectures requires a significantly richer model of parallelization than what is assumed in the classical [Gra66, GLLRK79] or even more recent [BB90, BB91, KM92, TWY92, WC92] efforts in that field. To the best of our knowledge, there have been no theoretical results in the literature on parallel task scheduling that consider multiple system resources and explore resource sharing among concurrent tasks, or study the implications of pipelined parallelism and data communication costs. This is an area of growing interest, however; in addition to our own effort, recently Shachnai and Turek [ST94] have independently obtained some results on multiresource parallel task scheduling. Their results on makespan scheduling are similar to ours although they assume a very different model of resource usage.

3 Problem Formulation

3.1 Definitions

We consider shared-nothing systems with *identical* multiprogrammed resource sites connected by an interconnection network. Each site is a collection of d system resources that are assumed to be *time-sliceable* or *preemptable*, in the sense that they can be time-shared among different operations at low overhead. Resources like the CPU(s), the disk(s), and the network interface(s) or communication processor(s) are preemptable, while memory is not.

An *operator tree* [GHK92, Hon92, Sch90] is created as a "macro-expansion" of an execution plan tree by refining each node into a subtree of physical operator nodes, e.g., `scan`, `probe`, `build` (Figure 1(a,b)). Edges represent the flow of data as well as two forms of timing constraints between operators: pipelining (thin edges) and blocking (thick edges). A *query task* is a maximal subgraph of the operator tree containing only pipelining edges. A *query task tree* is created from an operator tree by representing query tasks as single nodes (Figure 1(c)).

The above trees clarify the definitions of the three forms of intra-query parallelism:

- *Partitioned parallelism:* A single node of the operator tree is executed on a set of sites by appropriately partitioning its input data sets.

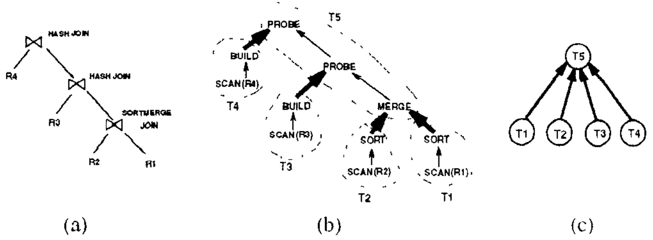


Figure 1: (a) An execution plan tree. (b) The corresponding operator tree. (c) The corresponding query task tree. The thick edges in (b) indicate blocking constraints.

- *Pipelined parallelism:* The operators of a single node of the task tree are executed on a set of sites in a pipelined manner.
- *Independent parallelism:* Nodes of the task tree with no path between them are executed on a set of sites independent of each other. For example, in Figure 1, tasks T1-T4 can be executed in parallel, whereas task T5 must await the completion of T1-T4.

The *home* of an operator is the set of sites allotted to its execution. Each operator is either *rooted*, if its home is fixed by data placement constraints (e.g., scanning the materialized result of a previous task), or *floating*, if the resource scheduler is free to determine its parallelization.

3.2 Overview

A *parallel schedule* consists of (1) an operator tree and (2) an allocation of system resources to operators. Given a query execution plan, our goal is to find a parallel schedule with minimal response time. To account for the communication overhead of parallelism, we initially restrict our attention to partitioned parallelism that is *coarse grain* [GW93, GY93]. That is, we ignore operator parallelizations whose ratio of computation costs to communication overhead is not sufficiently high, as most of them are bound to be ineffective.

Based on the above restriction, we devise an algorithm for scheduling bushy execution plan trees that consists of the following steps:

1. Construct the corresponding operator and task trees, and deterministically split the latter into synchronized phases [TL93], where each phase contains tasks with no (blocking) paths between them.
2. For each operator, determine its individual resource requirements using hardware parameters, DBMS statistics, and conventional optimizer cost models (e.g., [HCY94, SAC$^+$79]).
3. For each floating operator, determine the degree of coarse grain parallelism based on the relative cost of computation and communication (partitioned parallelism).
4. For each phase of the task tree, schedule all floating operators on the set of available sites using a multi-dimensional list scheduling heuristic that is *provably near-optimal* in the space of coarse grain parallel executions (pipelined and independent parallelism).

We then propose a technique for selecting an operator parallelization that allows us to relax the coarse granularity restriction (Step 3). Combining this technique with our list scheduling rule for independent operators results in an algorithm that is provably near-optimal in the space of *all possible* parallel executions.

3.3 Assumptions

Our approach is based on the following set of assumptions:

A1. No Memory Limitations. An operator is always allotted sufficient memory buffers to allow the execution of an operator pipeline to proceed in a single phase. For example, when executing a pipeline of `probe` operators, the hash tables built on the inner relations are assumed to be memory-resident. To the best of our knowledge, developing an accurate memory usage model for parallel query optimization is an open problem.

A2. No Time-Sharing Overhead. Following Ganguly et al. [GHK92], slicing a preemptable resource among multiple operators introduces no additional resource costs.

A3. Uniform Resource Usage. Following Ganguly et al. [GHK92], usage of a preemptable resource by an operator is uniformly spread over the execution of the operator.

A4. Non-increasing Operator Execution Times. For the range of coarse grain parallelism considered, an operator's execution time is a non-increasing function of its degree of parallelism, i.e., allotting more sites cannot increase its response time.

A5. Dynamically Repartitioned Pipelined Outputs. The output of an operator in a pipeline is always repartitioned to serve as input to the next one. This is almost always accurate, e.g., when the join attributes of pipelined joins are different, the degrees of partitioned parallelism differ, or different declustering schemes must be used for load balancing.

4 Coarse Grain Parallelization of Operators

4.1 A Resource Usage Model

Our treatment of resource usage is based on the model of preemptable resources proposed by Ganguly et al. [GHK92], which we briefly describe here. The usage of a single resource by an operator is modeled by two parameters, T and W, where T is the elapsed time after which the resource is freed (i.e., the response time of the operator) and W is the work measured as the effective time for which the resource is used by the operator. Intuitively, the resource is kept busy by the operator only W/T of the time. Although this abstraction can model the true utilization of a system resource, it does not allow us to predict exactly when the busy periods are. Thus, we make assumption A3 which, in conjunction with assumption A2, leads to straightforward quantification of the effects of resource sharing [GHK92].

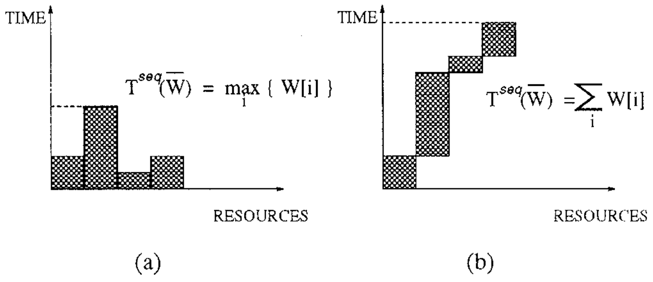


Figure 2: Extremes in usage of d-dimensional resource sites: (a) perfect overlap and (b) zero overlap.

We extend the model of Ganguly et al. [GHK92] and describe the usage by an isolated operator of a site comprising of d preemptable resources by the pair $(T^{seq}, \overline{W})$. Parameter T^{seq} is the (sequential) execution time of the operator, while $\overline{W}$ is a d-dimensional *work vector* whose components denote the work done on individual resources. Our model assumes a fixed numbering of system resources for all sites; for example, dimensions 1, 2, 3, and 4 may correspond to CPU, disk-1, disk-2, and network interface, respectively. Time T^{seq} is actually a function of the operator's individual resource requirements, i.e., its work vector $\overline{W}$ (sometimes emphasized by using $T^{seq}(\overline{W})$ instead of T^{seq}), and the amount of *overlap* that can be achieved between processing at different resources. This overlap is a system parameter that depends on the hardware and software architecture of the resource sites (e.g., buffering architecture for disk I/O) as well as the algorithm implementing the operator. An important constraint for T^{seq}, however, is that it can never be less than the amount of work done on any single resource and it can never exceed the total work performed. As shown in Figure 2[1], this is more formally expressed as

$$\max_{1 \leq i \leq d} \{W[i]\} \leq T^{seq}(\overline{W}) \leq \sum_{i=1}^{d} W[i].$$

4.2 Quantifying Coarse Grain Parallelism

As is well known, increasing the parallelism of an operator reduces its execution time until a saturation point is reached, beyond which additional parallelism causes a speed-down, due to excessive communication startup and coordination overhead over too many sites [DGS+90]. To avoid operating beyond that point, we need to ensure that the granules of the parallel execution are sufficienty coarse. In particular, in the spirit of Stone [Sto87], we define the *granularity* of a d-dimensional parallel operator op as the ratio $W_p(\mathrm{op})/W_c(\mathrm{op}, N)$, where

- $W_p(\mathrm{op})$ denotes the total amount of work performed during the execution of op on a single site, when all its operands are locally resident (i.e., zero communication cost); it corresponds to the *processing area* [GW93] of op and is constant for all possible executions of op; and

[1] Figure 2 is actually a little misleading since, by assumption A3, the work performed on any resource should be *uniformly* spread over T^{seq}.

- $W_c(\mathrm{op}, N)$ denotes the total communication overhead incurred when the execution of op is distributed across N sites; it corresponds to the *communication area* of a parallel execution of op on N sites and is a non-decreasing function of N.

Using the above notions, we extend earlier quantifications of coarse grain parallelism [GW93] to our multi-dimensional operator model as follows:

Definition 4.1 A parallel execution of an operator op on N resource sites is *coarse grain with parameter f* (referred to as a CG_f execution) if the communication area of the execution is no more than f times the processing area of op, that is, $W_c(\mathrm{op}, N) \le f\ W_p(\mathrm{op})$.

4.3 Degree of Partitioned Parallelism

Assuming zero communication costs, the resource requirements of the operator are described by a d-dimensional work vector $\overline{W}$ whose components can be derived from system parameters and traditional optimizer cost models. By definition, the processing area of the operator $W_p(\mathrm{op})$ is simply the sum of $\overline{W}$'s components, i.e., $W_p(\mathrm{op}) = \sum_{i=1}^{d} W[i]$.

Let D denote the total size (in bytes) of the operator's input and output data set(s) that are transferred over the interconnect. We use a simple model of communication costs in which the total communication overhead for the parallel execution of an operator on N sites is estimated as:

$$W_c(\mathrm{op}, N) = \alpha\ N + \beta\ D.$$

where α, β are architecture-specific parameters specified as follows:

- α is the startup cost for each participating site, and
- β is the time spent at the network interface and/or communication processor per unit of data transferred.

This model of operator communication costs is substantiated by the experimental results of DeWitt et al. on the Gamma shared-nothing database machine [DGS+90], and simpler forms of this model have been adopted in previous studies of shared-nothing systems [GMSY93, WFA92].

Note that the startup cost cannot, in general, be distributed among the participating sites. Rather, it is inherently serial and is incurred at a single site (the designated "coordinator" for the parallel execution). This implies that there always exists some degree of parallelism beyond which the startup overhead at the coordinator dominates the actual processing time.

The following proposition is an immediate consequence of Definition 4.1 and our communication cost model.

Proposition 4.1 The maximum allowable degree of intra-operator parallelism for a CG_f execution of operator op is denoted by $N_{max}(\mathrm{op}, f)$ and is determined by the formula

$$N_{max}(\mathrm{op}, f) = \max\{ \left\lfloor \frac{f\ W_p(\mathrm{op}) - \beta\ D}{\alpha} \right\rfloor\ ,\ 1\}$$

5 The Scheduling Algorithm

5.1 Notation

Table 1 summarizes the notation used in this section with a brief description of its semantics. Detailed definitions of some of these parameters are given below. Additional notation will be introduced when necessary. Vector $\overline{W}_{\mathrm{op}_i}$

Parameter	Semantics
P	Number of system sites
d	Site dimensionality (no. of resources per site)
s_j	System site ($j = 1, \dots, P$)
$work(s_j)$	Operator clones sharing site s_j
$T^{site}(s_j)$	Execution time for all operator clones at site s_j
M	Number of concurrent operators
op_i	Operator, e.g., scan, build ($i = 1, \dots, M$)
N_i	Degree of partitioned parallelism (number of clones) for op_i
$\overline{W}_{\mathrm{op}_i}$	Work vector for op_i (including communication costs for N_i sites)
$T^{par}(\mathrm{op}_i, N_i)$	Time of parallel execution of op_i on N_i sites while alone in system
$T^{seq}(\overline{W})$	Time of sequential execution of operator with resource requirements $\overline{W}$ (Section 4.1)
$l(\overline{W}), l(S)$	Length of a work vector $\overline{W}$ or set of work vectors S

Table 1: Notation

describes the total (i.e., processing and communication) resource requirements of op_i, given its degree of parallelism N_i. Using the notions of communication and processing area defined in Section 4, the above is expressed as

$$\sum_{k=1}^{d} W_{\mathrm{op}_i}[k] = W_p(\mathrm{op}_i) + W_c(\mathrm{op}_i, N_i).$$

The individual components of $\overline{W}_{\mathrm{op}_i}$ are computed using architectural parameters, database statistics, and our model for communication costs[2].

The *length of a d-dimensional vector* $\overline{W}$ is its maximum component. The *length of a set S of d-dimensional vectors* is the maximum component in the vector sum of all the vectors in S. More formally,

$$l(\overline{W}) = \max_{1 \le k \le d}\{W[k]\}\quad,\quad l(S) = \max_{1 \le k \le d}\{\sum_{\overline{W} \in S} W[k]\}.$$

5.2 Modeling Parallel Execution and Resource Sharing

In this section, we present a set of extensions to the (one-dimensional) cost model of a traditional DBMS based on the multi-dimensional resource usage formulation described in Section 4.1. Our extensions account for all forms of parallelism and quantify the effects of resource sharing on the response time of a parallel execution.

[2]The actual distribution of costs among the vector's components is immaterial as far as our model is concerned.

5.2.1 Partitioned Parallelism

In partitioned parallelism, the work vector of an operator is partitioned among a set of *operator clones* [GHK92]. Each clone executes on a single site and works on a portion of the operator's data. Consider an operator op_i that is distributed across N_i sites and runs in isolation, without experiencing resource contention. Partitioning $\overline{W}_{\mathrm{op}_i}$ into the work vectors for the operator clones is determined based on statistical information kept in the DBMS catalogs. Given such a partitioning $< \overline{W}_1, \overline{W}_2, \ldots, \overline{W}_{N_i} >$, where $\sum_{k=1}^{N_i} \overline{W}_k = \overline{W}_{\mathrm{op}_i}$, the parallel execution time for op_i can be expressed as the maximum of the sequential execution times of the N_i clones; that is,

$$T^{par}(\mathrm{op}_i, N_i) = \max_{1 \le k \le N_i} \{ T^{seq}(\overline{W}_k) \}. \qquad (1)$$

5.2.2 Pipelined and Independent Parallelism

Definition 5.1 Given a collection of M operators to be executed concurrently $\{\mathrm{op}_i, i = 1 \ldots M\}$ and their respective degrees of partitioned parallelism $\{N_i, i = 1 \ldots M\}$, a *schedule* is a mapping of the $\sum_{i=1}^{M} N_i$ operator clones to the set of available sites such that no two clones of the same operator are mapped to the same site.

The constraint on the mapping of operator clones to sites ensures that N_i is the true degree of parallelism for op_i so that Equation (1) is still valid.

The effects of time-sharing a site among many operators can be quantified as follows. Let $work(s_j)$ denote the set of all operator clones (or, equivalently, all work vectors) mapped to site s_j under a particular schedule. Since all resources are preemptable, the execution time for all the operator clones scheduled at s_j is determined by the ability to overlap the processing of resource requests by different operators. Specifically, under our model of preemptable resources described in Section 4.1, the execution time for all the operator clones scheduled at s_j is defined as

$$T^{site}(s_j) = \max\{ \max_{\overline{W} \in work(s_j)} \{T^{seq}(\overline{W})\} , l(work(s_j)) \}. \qquad (2)$$

For example, consider two 2-dimensional operator clones with resource usage pairs $(T_1^{seq}, \overline{W}_1)$ = (22,[10,15]) and $(T_2^{seq}, \overline{W}_2)$ = (10,[10,5]) placed at s_j. In this case, $\overline{W}_1 + \overline{W}_2$ = [20,20], which means that the total requirements of the two clones ($l(\{\overline{W}_1, \overline{W}_2\}) = 20$) can be "squeezed" into the response time of the first clone ($T_1^{seq} = 22$), i.e., $T^{site}(s_j) = 22$. On the other hand, consider $(T_1^{seq}, \overline{W}_1)$ placed at s_j with $(T_3^{seq}, \overline{W}_3)$ = (10,[5,10]). In this case, $\overline{W}_1 + \overline{W}_3$ = [15,25], and the second resource gets congested, i.e., $T^{site}(s_j) = l(\{\overline{W}_1, \overline{W}_3\}) = 25$, while $\max\{T_1^{seq}, T_3^{seq}\} = 22$.

Let SCHED be a schedule for the parallel execution of $\{\mathrm{op}_i, i = 1 \ldots M\}$ on a set of resource sites $\{s_j, j = 1 \ldots P\}$. Clearly, the response time of SCHED is determined by the most heavily loaded site. Thus, we can combine Equations (2) and (1) to estimate the response time as follows:

$$\begin{aligned} T^{par}(\mathrm{SCHED}, P) &= \max_{1 \le j \le P} \{ T^{site}(s_j) \} \\ &= \max\{ \max_{1 \le i \le M} \{T^{par}(\mathrm{op}_i, N_i)\} , \\ &\qquad \max_{1 \le j \le P} \{l(work(s_j))\} \}. \quad (3) \end{aligned}$$

Equation (3) defines the optimization metric for our scheduling algorithm, described in the next section. Intuitively the formula states that the response time of a parallel execution schedule is determined by either the slowest executing operator, or the load at the most heavily congested resource in the system, whichever is greater.

5.3 A Near-Optimal Heuristic for Independent Query Tasks

The *performance ratio* of a scheduling algorithm is defined as the ratio of the response time of the schedule it generates over that of the optimal schedule. In this section, we develop a heuristic for scheduling independent query tasks that is provably near-optimal, i.e., with a constant bound on the performance ratio. In Section 5.4, we address the general query task tree scheduling problem. A collection of independent query tasks (pipelines) is essentially a collection of operators that can be executed concurrently. Operators within each task form producer-consumer pairs that communicate across the interconnection network, whereas operators in different tasks are completely independent. More specifically, let R, F denote the set of all rooted and floating operators respectively, that is $R \cup F = \{\mathrm{op}_i, i = 1 \ldots M\}$. Let $N = \sum_{i=1}^{M} N_i$, where the degree of parallelism N_i is determined by Proposition 4.1 for $\mathrm{op}_i \in F$ and by the existing data placement constraints for $\mathrm{op}_i \in R$.

The parallel execution time of an individual rooted operator is fixed by its parallelization. By assumption A4 and the calculation of N_i, the parallel execution time of an individual floating operator is optimal in the space of CG_f executions. Hence, depending on the operator type, the left input of max in (3), i.e., $T^{par}(\mathrm{op}_i, N_i)$, is either fixed or minimized. Consequently, minimization of response time (equation (3)) translates to determining a mapping of the N work vectors obtained through the cloning of operators in $R \cup F$ to the P d-dimensional sites, such that

(A) no two vectors from the same operator are mapped to the same site,

(B) data placement constraints for rooted operators are satisfied, and

(C) the maximum resource usage among all system resources, i.e., the right input of max in (3), is minimized.

This is essentially an instance of the *d-dimensional bin-design* problem (the dual of the d-dimensional vector-packing problem) [CGJ84]. In vector-packing terminology, our scheduling problem may be stated as follows:

*Given a collection of positive d-dimensional vectors (the work vectors) and a set of P d-dimensional bins (the system sites), determine a **packing** of the vectors in the bins that obeys constraints **(A)** and **(B)** and minimizes the required common bin capacity (the maximum resource usage in the system).*

This problem is clearly $\mathcal{NP}$-hard since it reduces to traditional multiprocessor scheduling for $d = 1$, $R = \emptyset$, and $N_i = 1$ for all i. Given the intractability of the problem, we develop an approximation algorithm, OPERATORSCHEDULE , that runs in polynomial time and guarantees a constant bound on the performance ratio. OPERATORSCHEDULE belongs to the class of *list scheduling* algorithms originally proposed by Graham [Gra66]. The algorithm begins by placing the work vectors of all rooted operators at their respective sites and computing the degree of coarse grain parallelism for all floating operators. It then proceeds to schedule floating operators according to the following *list scheduling rule*: Consider the list of work vectors resulting from the cloning of all floating operators in non-increasing order of their maximum component; at each step, pack the next vector in the least filled allowable bin/site (that is, pack the vector in the site s_j such that $l(work(s_j))$ is minimal among all bins not containing other vectors of that operator). OPERATORSCHEDULE is depicted in Figure 3.

```
Input: A set of rooted and floating operators $R \cup F$, a set of $P$
    sites $\{s_1, \ldots, s_P\}$, and a granularity parameter $f$
Output: A schedule ($\{work(s_j), j = 1, \ldots, P\}$) for the CG$_f$
    execution of $R \cup F$ satisfying (A)-(B)

foreach $op_i \in R$ do
  place the work vectors of $op_i$ at their respective sites
end-for
for each $op_i \in F$ do
  set the degree of intra-operator parallelism
    $N_i = \min\{N_{max}(op_i, f), P\}$
  let $L_i =< \overline{w}_1, \ldots, \overline{w}_{N_i} >$ be the list of work
    vectors for $op_i$'s clones
end-for
let $L =< \overline{w}_1, \ldots, \overline{w}_N >$ be the list of all floating work
  vectors in non-increasing order of $l(\overline{w}_i)$
for $k = 1$ to $N$ do
  let $op_i$ be the operator whose cloning produced $\overline{w}_k$
  let $s$ be a site with $work(s) \cap L_i = \emptyset$ such that
    $l(work(s)) = \min_{s_j,\ work(s_j) \cap L_i = \emptyset}\{l(work(s_j))\}$
  set $work(s) = work(s) \cup \{\overline{w}_k\}$
end-for
```

Figure 3: The OPERATORSCHEDULE algorithm

The following theorem bounds the worst-case performance ratio of our algorithm. As with all theoretical results presented here, the theorem is stated without proof due to space constraints. The details can be found in the full version of this paper [GI96].

Theorem 5.1 The parallel execution time of the schedule returned by OPERATORSCHEDULE is

(a) within $(2d + 1)$ of the length of the optimal schedule that uses the same degrees of intra-operator parallelism for all floating operators, and

(b) within $(2d(fd + 1) + 1)$ of the optimal CG$_f$ schedule length.

We also provide an upper bound on the asymptotic time complexity of OPERATORSCHEDULE.

Proposition 5.1 OPERATORSCHEDULE runs in time $O(MP(M + \log P))$, where M is the number of concurrent operators and P is the number of system sites.

5.4 Handling Data Dependencies

Scheduling arbitrary query task trees must ensure that the blocking constraints specified by the tree's edges are satisfied. For this, we split a query task tree into synchronized phases or "shelves" [NSHL95, TL93]. Each phase contains independent tasks that are to be executed concurrently, after the completion of all tasks in the previous phase. The number of phases is equal to the height of the task tree and each task is scheduled in the phase closest to the root that does not violate the precedence constraints. For example, the plan in Figure 1 is executed in two distinct phases containing tasks T1-T4 and task T5, respectively. This corresponds to the $MinShelf$ policy of Tan and Lu [TL93]. Resource scheduling within each phase is performed by the OPERATORSCHEDULE algorithm. The full algorithm, TREESCHEDULE is depicted in Figure 4.

```
Input: A query task tree $T = (V, E)$, a set of $P$ sites $\{s_1, \ldots, s_P\}$,
    and a granularity parameter $f$
Output: A schedule for the CG$_f$ execution of $T$

for i = $height(T)$ downto 0 do
  $OP = \emptyset$
  foreach node $v \in V$ such that $level(v) = i$ do
    $OP = OP \cup$ {operators in task $v$}
  end-for
  call OPERATORSCHEDULE( $OP$, $\{s_1, \ldots, s_P\}$, $f$)
end-for
```

Figure 4: The TREESCHEDULE algorithm

Observe that for any query execution plan the number of nodes in the operator tree is bounded by a small constant times the number of joins in the query, e.g., expanding a hash-join gives at most four operator nodes. Combining this observation with Proposition 5.1 gives the following complexity bound for TREESCHEDULE .

Proposition 5.2 TREESCHEDULE runs in time $O(JP(J + \log P))$, where J is the number of nodes in the query execution plan and P is the number of system sites.

5.5 Comments on the Effectiveness of the Heuristics

Theorem 5.1 derives an upper bound on the worst-case performance ratio of the OPERATORSCHEDULE algorithm for scheduling a collection of CG$_f$ concurrent operators. In general, the expected output quality of our heuristic should be much better than the worst-case bounds, especially for a set of operators with a good "mix" of resource requirements. This conjecture is supported by theoretical results on the expected performance of vector packing [KLMS84]. The big advantage of OPERATORSCHEDULE compared to previous approaches is its ability to explore resource sharing possibilities and balance the resource workloads at individual sites.

Deriving performance bounds for the schedule produced by the TREESCHEDULE algorithm is a much more difficult problem. Theorem 5.1 ensures that scheduling within each phase is near-optimal *given its data placement constraints.* When scheduling a query task tree, the scheduling decisions made at earlier phases may impose data placement constraints on the phases that follow. For example, the `build` and `probe` operators of a hash join belong to two adjacent phases because of their sequential dependency (the hash table has to be complete before probing can begin). Furthermore, the `probe` operator has to be executed at the set of sites that hold the hash table, that is, the home of the `build`. Such interdependencies between phases complicate any proof of suboptimality bounds for the TREESCHEDULE algorithm. At this point, we have not been able to obtain theoretical results on the quality of the schedule produced for the entire query task tree. However, given the load balancing capabilities of the OPERATORSCHEDULE algorithm, we feel confident that TREESCHEDULE will outperform previous approaches. Our conjectures for both OPERATORSCHEDULE and TREESCHEDULE are supported by the results of a preliminary experimental evaluation presented in the next section.

6 Experimental Performance Evaluation

In this section, we describe the results of several experiments we have conducted comparing the average performance of our multi-dimensional scheduling algorithm with a one-dimensional "synchronous execution time" algorithm that we developed based on previous work [HCY94, LCRY93]. Another point of interest is examining how close the response time of the generated schedule is to that of the optimal coarse grain schedule *on the average*. We start by presenting our experimental testbed and methodology.

6.1 Experimental Testbed

We have experimented with the following algorithms:

- SYNCHRONOUS : Combination of the synchronous execution time method of Hsiao et al. [HCY94] for processor allocation for independent parallelism with the two-phase minimax technique of Lo et al. [LCRY93] for *optimally* distributing processors across the stages of a hash-join pipeline. Although these strategies were originally proposed for shared-disk systems, they were appropriately extended to account for the data redistribution costs in a shared-nothing environment.
- TREESCHEDULE : Multi-dimensional list scheduling in synchronized phases.
- OPTBOUND : Hypothetical algorithm achieving a lower bound on the optimal response time.

We selected SYNCHRONOUS as a one-dimensional adversary since it is the "state-of-the-art" method for exploiting bushy tree parallelism in parallel query execution[3] [WFA95]. Prior research has demonstrated the advantages offered by such parallelism, especially for large queries [CYW92]. To the best of our knowledge, optimal processor distribution within general join pipelines remains an open problem. We therefore decided to restrict our experiments to bushy hash-join query plans so that the optimal technique of Lo et al. could be used in SYNCHRONOUS. We should stress, however, that TREESCHEDULE is a general query scheduling algorithm that can be applied to *any* bushy plan.

Some additional assumptions were made to obtain a specific experimental model from the general parallel execution model described in Sections 4 and 5:

EA1. No Execution Skew: With the exception of startup cost, the work vector of an operator is distributed perfectly among all sites participating in its execution. Startup is added to only one of these sites, the "coordinator site" for the parallel execution, and is equally divided between the coordinator's CPU and its network interface.

EA2. Uniform Resource Overlapping: The amount of overlap achieved between processing at different resources at a site can be characterized by a single system-wide parameter $\epsilon \in [0,1]$ for all query operators. This parameter allows us to express the response time of a work vector as a convex combination of the maximum and the sum of the vector components (see Section 4.1), i.e., $T(\overline{W}) = \epsilon(\max_{1\leq i\leq d}\{W[i]\}) + (1-\epsilon)\sum_{i=1}^{d} W[i]$. Small values of ϵ imply limited overlap, whereas values closer to 1 imply a larger degree of overlap. In the extreme cases, $\epsilon = 1$ gives $T(\overline{W}) = \max_{1\leq i\leq d}\{W[i]\}$ (perfect overlap), and $\epsilon = 0$ gives $T(\overline{W}) = \sum_{i=1}^{d} W[i]$ (zero overlap).

Finally, special precautions were taken to ensure that assumption A4 is not violated for any given value of the granularity parameter f. For each query operator, there exists an optimal degree of partitioned parallelism that minimizes the response time [WFA92], and beyond which startup costs will cause a speed-down. Our implementation makes sure that this optimal degree of parallelism is never exceeded for any operator.

We experimented with tree queries of 10, 20, 30, 40, and 50 joins. For each query size, twenty query graphs (trees) were randomly generated and for each graph a bushy execution plan was randomly selected. We assumed simple key join operations in which the size of the result relation is always equal to the size of the largest of the two join operands. The

[3]The *Fully Parallel Execution Method* [WFA95] applies only to main-memory parallel database systems.

comparison metric was the *average response times* of the schedules produced by the algorithms over all queries of the same size. Experiments were conducted with the resource overlap parameter ϵ varying between 10% and 70% and the granularity parameter f varying between 0.3 and 0.9. (The results presented in the next section are indicative of the results obtained for all values of ϵ and f.)

In all experiments, we assumed a system consisting of 3-dimensional sites with one CPU, one disk unit, and one network interface. The work vector components for the CPU and the disk were estimated using the cost model equations given by Hsiao et al. [HCY94]. The communication costs were calculated using the model described in Section 4.2. The values of the cost model parameters were obtained from the literature [GW93, HCY94, WFA92] and are summarized in Table 2[4].

Configuration/Catalog Parameters	Value
Number of Sites	10 - 140
CPU Speed	1 MIPS
Effective Disk Service Time per page	20 msec
Startup Cost per site (α)	15 msec
Network Transfer Cost per byte (β)	0.6 μsec
Tuple Size	128 bytes
Page Size	40 tuples
Relation Size	10^3 - 10^5 tuples
CPU Cost Parameters	**No. of Instr.**
Read Page from Disk	5000
Write Page to Disk	5000
Extract Tuple	300
Hash Tuple	100
Probe Hash Table	200

Table 2: Experiment Parameter Settings

6.2 Experimental Results

The first set of experiments studied the effect of different values of the granularity parameter f on the performance of TREESCHEDULE compared to that of SYNCHRONOUS (which is, of course, not affected by different values of f). The results for queries of 40 joins and a resource overlap of 30% (i.e., $\epsilon = 0.3$) are depicted in Figure 5(a). Clearly, for small values of f the coarse granularity condition is too restrictive, not allowing the execution system to fully exploit the available parallelism. As the value of f increases, the average plan response time drops substantially until the bound on operator parallelism is reached. As expected, the advantages of resource sharing are most evident for resource-limited situations (i.e., small parallel systems). Nevertheless, for sufficiently large values of f, our algorithm outperformed its one-dimensional adversary in the entire range of system and query sizes.

The second set of experiments studied the effect of the resource overlap parameter ϵ on the performance of the two algorithms, while the granularity parameter was kept constant. The performance results shown in Figure 5(b) (for queries of 40 joins) demonstrate that TREESCHEDULE consistently outperformed the SYNCHRONOUS algorithm for various values of f. Clearly, the benefits of multi-dimensional scheduling are more significant for smaller values of the overlap parameter. The reason is that lower overlap results in longer idle periods for the individual resources which our algorithm can exploit through time-sharing with other operations.

The average performance of the two scheduling algorithms for different query sizes is depicted in Figure 6(a) for two different system sizes (20 and 80 sites) and overlap $\epsilon = 0.5$. For TREESCHEDULE we assume f to be fixed at 0.7. Note that, for a given system size, the relative improvement obtained with TREESCHEDULE increases monotonically with the query size.

We should also mention that the asymptotic time complexity of SYNCHRONOUS is $O(JP\log(JP))$, where J is the number of joins in the query and P is the number of sites [LCRY93]. Thus, TREESCHEDULE appears to be slightly more expensive than SYNCHRONOUS, being quadratic in the size of the query (Proposition 5.2). We believe that this is a small price to pay compared to the significant performance improvement offered by resource sharing, especially for large queries and/or resource-limited situations.

For our final set of experiments, we examined the average performance of TREESCHEDULE compared to a lower bound on the response time of the optimal CG_f execution for a constant value of f. This lower bound, OPTBOUND, was estimated using the formula

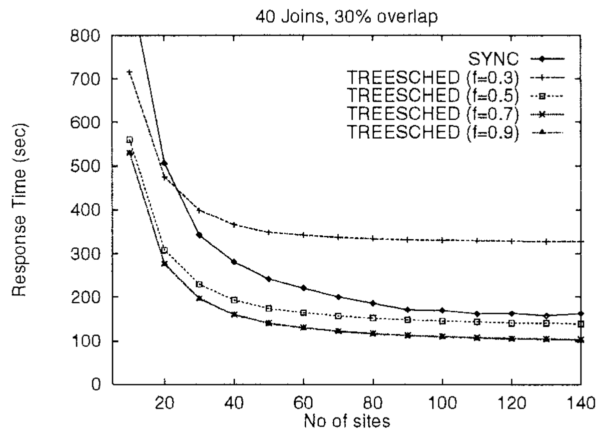
$$\text{OPTBOUND} = \max\{\ \frac{l(S)}{P}\ ,\ T(\text{CP})\ \},$$

where

- S is the set of work vectors for all operators in the query execution plan *assuming zero communication costs for each operator*, and
- $T(\text{CP})$ is the total response time of the critical (i.e., most time-consuming) path in the plan *assuming the maximum allowable degree of coarse grain parallelism for each operator*.

By assumption A4, OPTBOUND is indeed a lower bound on the length of the optimal CG_f execution [GI96]. The results for queries of 20 and 40 joins are shown in Figure 6(b) for $f = 0.7$ and overlap $\epsilon = 0.5$. These curves verified our expectations, showing that the average performance of TREESCHEDULE is much closer to optimal than what we would expect from the worst-case bound derived in Theorem 5.1 for each plan phase. These results are in accordance with the theoretical results of Karp et al. [KLMS84] who used a probabilistic model to prove that even very simple vector-packing heuristics can be expected to produce packings in which very little of the capacity of the bins is wasted.

[4]The CPU speed and disk service rate were chosen so that the system is relatively balanced (i.e., not heavily CPU or IO bound).



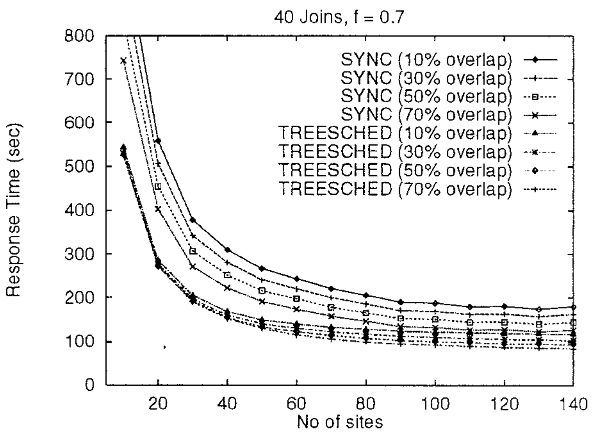
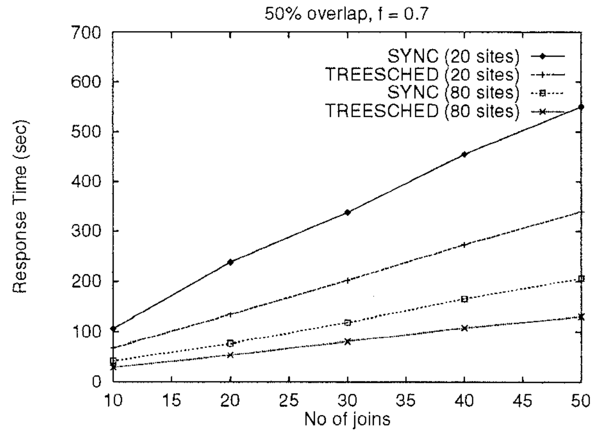


Figure 5: (a) Effect of the granularity parameter (f). (b) Effect of the resource overlap parameter (ϵ).



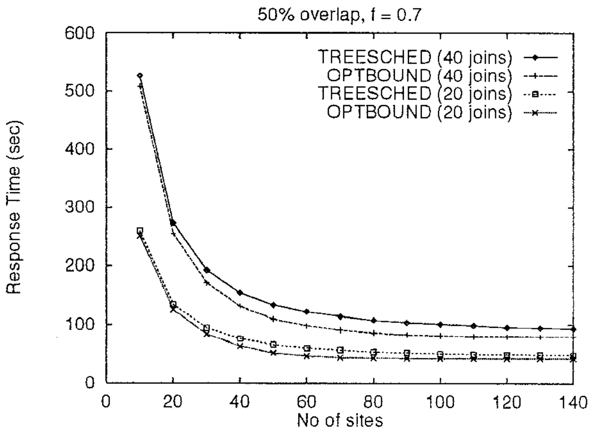


Figure 6: (a) Effect of query size. (b) Average Performance of TREESCHEDULE vs. Optimal.

7 Extensions for Malleable Operators

In this section, we extend our list scheduling technique to handle the more general *malleable* scheduling problem. The degree of parallelism for the floating query operators is no longer determined through a coarse granularity condition. Instead, floating operators are malleable, in the sense that the scheduler is free to determine their parallelization so that the execution time is minimized over all possible parallel schedules. Since rooted operators have no effect on the quality of the generated schedule (their scheduling is determined by data placement constraints) we will only consider floating operators in this section.

Let $\underline{N} = (N_1, \ldots, N_M)$ denote a parallelization (i.e., the degrees of parallelism) of a given set of independent operators, and let $S(\underline{N}) = (\overline{W}_{\text{op}_1}(N_1), \ldots, \overline{W}_{\text{op}_M}(N_M))$ be the set of (total) work vectors for the operators (including the communication costs for the given parallelization). Finally, define $h(\underline{N}) = \max_{1 \le i \le M}\{T^{par}(\text{op}_i, N_i)\}$, i.e., the parallel execution time of the slowest operator. In proving the $(2d+1)$ suboptimality bound for OPERATORSCHEDULE [GI96] we actually show that the makespan of the schedule produced by our list scheduling rule for any given operator parallelization $\underline{N}$ satisfies the following inequality:

$$T^{par}(\text{SCHED}, P, \underline{N}) \le (2d+1)\max\{\frac{l(S(\underline{N}))}{P}, h(\underline{N})\},$$

where $LB(\underline{N}) = \max\{\frac{l(S(\underline{N}))}{P}, h(\underline{N})\}$ is a lower bound on the optimal response time for the given parallelization.

Our goal is to determine a particular operator parallelization $\underline{N}$ such that when $\underline{N}$ is used as input to our list scheduling technique the resulting schedule is guaranteed to be within $(2d+1)$ of the optimal schedule (over all possible parallelizations). The following lemma formalizes our expectations.

Lemma 7.1 Let $\underline{N}^*$ denote the parallelization of operators in the optimal execution schedule. Let $\underline{N}$ be another (possibly identical) parallelization such that $LB(\underline{N}) \le LB(\underline{N}^*)$. Then, applying our list scheduling rule to $\underline{N}$ will return an execution schedule whose length is within $(2d+1)$ of the optimal schedule length.

We now present a greedy selection algorithm for generating a family of parallelizations. The algorithm is an adaptation of the **GF** method presented by Turek et al. [TWY92] based on the observation that in our work vector model, for any operator op, if $n \le m$ then $\overline{W}_{\text{op}}(n) \le_d \overline{W}_{\text{op}}(m)$[5] :

[5] $\le_d$ stands for componentwise less-than, i.e., $\overline{w}_1 \le_d \overline{w}_2$ iff $\overline{w}_1[i] \le \overline{w}_2[i]$ for $i = 1, \ldots, d$.

1. The first candidate parallelization is the minimum total work parallelization $\underline{N}^1 = (1, 1, \dots, 1)$.
2. The k^{th} candidate parallelization is determined by the $(k-1)^{th}$ parallelization by first finding the operator whose execution time is equal to $h(\underline{N}^{k-1})$ and increasing its degree of parallelism by one.
3. The algorithm terminates when no more sites can be allotted to the largest operator.

Lemma 7.2 Let $\underline{N}^*$ denote the parallelization of operators in the optimal execution schedule. The above algorithm produces at least one parallelization $\underline{N}$ such that the following two properties hold:

1. $T^{par}(\text{op}_i, N_i) \leq h(\underline{N}^*)$ for all i, and
2. $\overline{W}_{\text{op}_i}(N_i) \leq_d \overline{W}_{\text{op}_i}(N_i^*)$ for all i.

From Lemma 7.2 and the definition of the lower bound $LB()$, at least one of the operator parallelizations produced by the algorithm will satisfy the conditions of Lemma 7.1.

Theorem 7.1 Let A be the family of parallelizations generated and let $\underline{N} \in A$ such that $LB(\underline{N}) = \min_{\underline{K} \in A}\{LB(\underline{K})\}$. Then, the schedule generated by our list scheduling rule for the parallelization $\underline{N}$ is within $(2d+1)$ of the optimal parallel schedule length.

The number of parallelizations generated by our algorithm is bounded by $1 + M(P-1)$ and so the complexity of selecting an operator parallelization is $O(MP \log M)$. Thus, this preprocessing step does not affect the asymptotic complexity of our scheduler. Also note that Theorem 7.1 does not depend on the non-increasing execution times assumption (A4) or any particular model for communication costs. The only assumption required is that of non-decreasing work vectors.

8 Conclusions

In this paper, we have addressed the open problem of multi-dimensional resource scheduling for complex queries in parallel database systems. Our approach is based on (1) a model of resource usage that allows the scheduler to explore the possibilities for resource sharing among concurrent operations and quantify the effects of this sharing on the parallel execution time, and (2) a quantification of the notion of coarse grain parallelism for query plan operators. Using these tools we developed a vector-packing formulation of the resource scheduling problem for independent query tasks, and proposed OPERATORSCHEDULE, a fast list scheduling heuristic that is provably near-optimal in the class of coarse grain executions. We then extended our approach to handle the blocking constraints in a bushy query plan by splitting its execution into synchronized phases. The resulting algorithm, TREESCHEDULE, exploits all forms of intra-query parallelism and allows effective resource sharing among operators executing concurrently. We also verified the effectiveness of our scheduling methods compared to both previous (one-dimensional) approaches and the optimal solution through a series of experimental results. Finally, we proposed a technique that allows us to relax the coarse granularity restriction and obtain a provably near-optimal list scheduling method for the malleable independent operator scheduling problem. In practice, the coarse granularity condition provides a fast way of determining an efficient parallelization based on system parameters. The more sophisticated greedy selection technique can be used when the additional scheduling overhead is justified.

The multi-dimensional model of query parallelization and resource scheduling proposed in this paper suggests several directions for future research. First, the framework is useful only for resources that are preemptable. Incorporating non-preemptable resources such as memory requires an even richer model of parallelization and thus remains an open question. Memory, in particular, introduces an additional level of complexity since the amount of work performed by an operator often depends on the amount of available memory. Second, the assumption of zero time-sharing overhead (A2) may not be accurate for certain types of resources. For example, disks do not time share as gracefully as processors or network interfaces; slicing a disk among many tasks can reduce the disk's effective bandwidth. Extending our model and algorithms to consider different degrees of "preemptability" for system resources is a challenging issue. Finally, given the generality of our scheduling framework, it would be interesting to investigate its applicability to other "multi-dimensional processing" situations (e.g., request scheduling in multimedia storage servers). These questions form the basis of our current and future research.

References

[BB90] K. P. Belkhale and P. Banerjee. "Approximate Algorithms for the Partitionable Independent Task Scheduling Problem". In *Proc. of the 1990 Intl. Conference on Parallel Processing*, August 1990.

[BB91] K. P. Belkhale and P. Banerjee. "A Scheduling Algorithm for Parallelizable Dependent Tasks". In *Proc. of the 5th Intl. Parallel Processing Symposium*, 1991.

[CGJ84] E.G. Coffman, Jr., M.R. Garey, and D.S. Johnson. "Approximation Algorithms for Bin-Packing – An Updated Survey". In *"Algorithm Design for Computing System Design"*. Springer-Verlag, New York, 1984.

[CHM95] C. Chekuri, W. Hasan, and R. Motwani. "Scheduling Problems in Parallel Query Optimization". In *Proc. of the 14th ACM Symposium on Principles of Database Systems*, San Jose, California, May 1995.

[CLYY92] M.-S. Chen, M.-L. Lo, P. S. Yu, and H. C. Young. "Using Segmented Right-Deep Trees for the Execution of Pipelined Hash Joins". In *Proc. of the 18th Intl. Conference on Very Large Data Bases*, Vancouver, Canada, August 1992.

[CYW92] M.-S. Chen, P. S. Yu, and K.-L. Wu. "Scheduling and Processor Allocation for Parallel Execution of Multi-Join Queries". In *Proc. of the 8th Intl. Conference on Data Engineering*, Phoenix, Arizona, February 1992.

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

[DGS+90] D. J. DeWitt, S. Ghandeharizadeh, D. A. Schneider, A. Bricker, H.-I Hsiao, and R. Rasmussen. "The Gamma Database Machine Project". *IEEE Transactions on Knowledge and Data Engineering*, 2(1), March 1990.

[DL89] J. Du and J. Y-T. Leung. "Complexity of Scheduling Parallel Task Systems". *SIAM Journal on Discrete Mathematics*, 2(4), November 1989.

[GGW95] S. Ganguly, A. Gerasoulis, and W. Wang. "Partitioning Pipelines with Communication Costs". In *Proc. of the 6th Intl. Conference on Information Systems and Data Management (CISMOD'95)*, Bombay, India, November 1995.

[GHK92] S. Ganguly, W. Hasan, and R. Krishnamurthy. "Query Optimization for Parallel Execution". In *Proc. of the 1992 ACM SIGMOD Intl. Conference on Management of Data*, San Diego, California, June 1992.

[GI96] M. N. Garofalakis and Y. E. Ioannidis. "Multi-dimensional Resource Scheduling for Parallel Queries". Unpublished manuscript, March 1996.

[GLLRK79] 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, 1979.

[GMSY93] S. Ghandeharizadeh, R. R. Meyer, G. L. Schultz, and J. Yackel. "Optimal Balanced Assignments and a Parallel Database Application". *ORSA Journal on Computing*, 5(2), Spring 1993.

[Gra66] R.L. Graham. "Bounds for Certain Multiprocessing Anomalies". *The Bell System Technical Journal*, 45, November 1966.

[GW93] S. Ganguly and W. Wang. "Optimizing Queries for Coarse Grain Parallelism". Technical Report LCSR-TR-218, Dept. of Computer Sciences, Rutgers University, October 1993.

[GY93] A. Gerasoulis and T. Yang. "On the Granularity and Clustering of Directed Acyclic Task Graphs". *IEEE Transactions on Parallel and Distributed Systems*, 4(6), June 1993.

[HCY94] H.-I Hsiao, M.-S. Chen, and P. S. Yu. "On Parallel Execution of Multiple Pipelined Hash Joins". In *Proc. of the 1994 ACM SIGMOD Intl. Conference on Management of Data*, Minneapolis, Minnesota, May 1994.

[HM94] W. Hasan and R. Motwani. "Optimization Algorithms for Exploiting the Parallelism-Communication Trade-off in Pipelined Parallelism". In *Proc. of the 20th Intl. Conference on Very Large Data Bases*, Santiago, Chile, August 1994.

[Hon92] W. Hong. "Exploiting Inter-Operation Parallelism in XPRS". In *Proc. of the 1992 ACM SIGMOD Intl. Conference on Management of Data*, San Diego, California, June 1992.

[KLMS84] R. M. Karp, M. Luby, and A. Marchetti-Spaccamela. "A Probabilistic Analysis of Multidimensional Bin Packing Problems". In *Proc. of the Annual ACM Symposium on the Theory of Computing*, 1984.

[KM92] R. Krishnamurti and E. Ma. "An Approximation Algorithm for Scheduling Tasks on Varying Partition Sizes in Partitionable Multiprocessor Systems". *IEEE Transactions on Computers*, 41(12), December 1992.

[LCRY93] M.-L. Lo, M.-S. Chen, C.V. Ravishankar, and P. S. Yu. "On Optimal Processor Allocation to Support Pipelined Hash Joins". In *Proc. of the 1993 ACM SIGMOD Intl. Conference on Management of Data*, Washington, D.C., June 1993.

[MD93] M. Mehta and D. J. DeWitt. "Dynamic Memory Allocation for Multiple-Query Workloads". In *Proc. of the 19th Intl. Conference on Very Large Data Bases*, Dublin, Ireland, 1993.

[MD95] M. Mehta and D. J. DeWitt. "Managing Intra-operator Parallelism in Parallel Database Systems". In *Proc. of the 21st Intl. Conference on Very Large Data Bases*, Zurich, Switzerland, September 1995.

[NSHL95] T. M. Niccum, J. Srivastava, B. Himatsingka, and J. Li. "Query Optimization and Processing in Parallel Databases". *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 22, 1995.

[RM95] E. Rahm and R. Marek. "Dynamic Multi-Resource Load Balancing in Parallel Database Systems". In *Proc. of the 21st Intl. Conference on Very Large Data Bases*, Zurich, Switzerland, September 1995.

[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 *Proc. of the 1979 ACM SIGMOD Intl. Conference on Management of Data*, Boston, Massachusetts, June 1979.

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

[ST94] H. Shachnai, and J. J. Turek. "Multiresource Malleable Task Scheduling". Submitted for publication, July 1994.

[Sto87] H. S. Stone. *"High-performance Computer Architecture"*. Reading, Mass. : Addison-Wesley Pub. Co., 1987.

[TL93] K.-L. Tan and H. Lu. "On Resource Scheduling of Multi-join Queries in Parallel Database Systems". *Information Processing Letters*, 48, 1993.

[TWY92] J. Turek, J. L. Wolf, and P. S. Yu. "Approximate Algorithms for Scheduling Parallelizable Tasks". In *Proc. of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures*, San Diego, California, June 1992.

[WC92] Q. Wang and K. H. Cheng. "A Heuristic of Scheduling Parallel Tasks and its Analysis". *SIAM Journal on Computing*, 21(2), April 1992.

[WFA92] A. N. Wilschut, J. Flokstra, and P. M.G. Apers. "Parallelism in a Main-Memory DBMS: The Performance of PRISMA/DB". In *Proc. of the 18th Intl. Conference on Very Large Data Bases*, Vancouver, Canada, August 1992.

[WFA95] A. N. Wilschut, J. Flokstra, and P. M.G. Apers. "Parallel Evaluation of Multi-join Queries". In *Proc. of the 1995 ACM SIGMOD Intl. Conference on Management of Data*, San Jose, California, May 1995.