Distributing a Database for Parallel Processing is NP-hard

H.C. Du

Department of Computer Science
University of Minnesota

Minneapolis, Minnesota 55455

ABSTRACT : In a database the response time to a query can be reduced if certain concurrent operations are provided. In order to maximize the degree of concurrency, data has to be carefully allocated. The complexities of the following two problems are studied in this paper : one is to allocate a database onto a multiple-disk system which maximizes the disk access concurrency; the other one is to allocate a database over a network which minimize the number of nodes being used and a complete parallel searching is possible for a set of queries. Both are shown to be NP-hard (computational intractable) problems.

Keywords : database, parallel Processing, NP-hard

A database (file) F is a collection of records and a record r is an ordered k-tuple $(r_1, r_2, \ldots, r_k)$. For each $1 \le i \le k$, $r_i \in D_i$, where D_i is the i-th attribute domain. Therefore, a database F is a subset of $D_1 X D_2 X \ldots X D_k$.

A retrieval request performed on the database is called a query. In order to respond to a query, a set of records (relevant records) from possibly millions of records needs to be accessed. A directory, which is a set of indexes, is usually constructed to help identify those relevant records. The time required to access all relevant records depends on the data allocation. By data allocation we mean how all records are stored on storage devices.

For example, if a file is large (this is a usual case) it must be stored on a secondary storage device such as a magnetic disk unit. In such a case, due to the physical limitation of a disk unit, the time required to access all relevant records can be measured in terms of the number of disk accesses issued. The number of disk accesses must be issued is equivalent to the number of buckets (pages) which contain at least one relevant record, if the directory access time is ignored.

In the past two decades, most effort related to designing an efficient file structure has been devoted to constructing an efficient directory and little attention has been paid to data allocation. However, there are some exceptional cases. In the previous example in order to minimize the time required to access all relevant records, similar records should be partitioned into the same buckets to minimize the number of buckets which contain at least one relevant record. By similar records we mean records that have high probability of being accessed together in response to a query.

Another approach to minimize the response time is to allow certain operations to be executed concurrently. We say that "a data allocation scheme is optimal" if maximum concurrency is achieved. Two examples will be considered here. One allows a database to be stored on a multiple disk system ; the other allows a database to be distributed over a network. These two problems are interesting and have been studied before. Our main objective in this paper is to determine the computational complexities of these two problems. We shall first introduce these two problems, and then show that they are NP-hard (computational intractable).

Problem 1. Assume that a file is stored on a set of m (m>1) independently accessible disks. The time required to access i buckets is becoming $\lceil i/m \rceil$ if all i buckets are uniformly distributed among m disks. The response time to a query is no longer proportional to the total number of buckets to be examined, but becomes proportional to the maximum number of buckets which need to be examined on a particular disk unit. Du and Sobolewski have studied the problem of finding an "optimal" allocation to assign all buckets in a file into m disks such that the maximum disk access concurrency can be achieved in response to a partial match query [1]. A partial match query is a query with certain attributes specified and the rest of them unspecified. A similar problem for binary files has also been studied by Du [2].

A more generalized problem for an arbitrary type of queries can be described as follows : A file F' can be considered as a collection of buckets and each bucket contains

no more than b records, where b is the bucket size. A set of n queries $q_1, q_2, \ldots, q_n$ and a response list $R(q_j)$ for each query q_j are given. Each response list $R(q_j) \subseteq F$ is a collection of buckets. Let P_j denote the set of buckets assigned to disk j, for $1 \leq j \leq m$. For each query q_j, let $<n_{j1}, n_{j2}, \ldots, n_{jm}>$ denote the distribution of all buckets in $R(q_j)$, where n_{ji} is the number of buckets in $R(q_j)$ which are assigned to disk unit i. Let T_i denote the response time to query q_i, for $1 \leq i \leq n$. Then $T_i = \max \{n_{i1}, n_{i2}, \ldots, n_{im}\}$. If we are interested in the worst case performance, the objective is to find an allocation such that $\max\{T_1, T_2, \ldots, T_n\}$ is minimized. If we are interested in the average case performance, the objective is to minimize $\sum_{i=1}^{n} T_i$ instead. Here we will only be concerned with optimal allocations in terms of the worst case performance.

In Problem 1 the basic units for allocation are records and the cases where a record may spread over several buckets are not considered. In the next problem, the basic units for allocation are attributes. Therefore, a record is partitioned into several parts and each part is allocated onto a node of a network.

Problem 2. A database (especially a relational database) can be considered as a table. Each row in the table is a k-tuple and each field in a k-tuple is an attribute. Each query type in the database may involve only some attributes and not all k attributes. Given a computer network and a set of query types, Ghosh has studied the problem of distributing a database (actually k attributes) over a network such that a request for each query type can be directed to different nodes for searching in parallel [4]. Later Srinivasan and Sankar [5] modified Ghosh's approximation algorithm by replacing the matrix multiplication over integer domain with the logical ADDing of some ZERO, ONE vectors. Thus, although the complexity of the new algorithm remains the same, the average running time is reduced. The problem they studied can be described as follows : A database of k attributes $\{A_1, A_2, \ldots, A_k\}$ and a set of n query types $\{q_1, q_2, \ldots, q_n\}$ are given. For each query type q_i, a set of attributes $R(q_i)$ which need to be retrieved by q_i is also given. The distribution problem for parallel searching is to determine the minimum

number m of nodes in the network such that i) all k attributes can be partitioned into $P_1, P_2, \ldots, P_m$ disjoint subsets and ii) there is at most one attribute in $R(q_i) \cap P_j$, for $1 \le i \le n$ and $1 \le j \le m$.

All the approaches taken in the previous studies to solve the above two problems are heuristic. However, the computational complexities of these two problems are still unknown. The main purpose of this paper is to show that they are NP-hard, hence computational intractable.

Theorem. Let F denote a set of objects and R_i a subset of F. Given F, a set $\{R_1, R_2, \ldots, R_n\}$, and a positive integer m, the problem of partitioning all objects in F into m subsets $P_1, P_2, \ldots, P_m$ such that $|R_i \cap P_j| \le 1$, for $1 \le i \le n$ and $1 \le j \le m$ is NP-complete, where $|X|$ denotes the cardinality of set X.

Proof : Since the problem can be easily solved on a nondeterministic Turing machine in polynomial time, it is clear that it is NP.

To show NP-hardness, we reduce the graph coloring problem, which is a well known NP-complete problem, to this problem.

The graph coloring problem (of which the "four coloring problem" is a special case) can be stated as follows : Given a graph G and a positive integer m, color all vertices in G by using m colors so that no two adjacent vertices are assigned the same color.

Let G=(V,E) be a given graph. V is the set of vertices and $E=\{e_1, e_2, \ldots, e_n\}$ is the set of edges, where e_i is the edge connecting vertices v_{i1} and v_{i2}.

Let F=V be the corresponding set of objects (thus each object in F corresponds to a vertex in V). For $1 \le i \le n$, let $R_i=\{v_{i1}, v_{i2}\}$, where vertices v_{i1} and v_{i2} are connected by edge e_i.

Let the action of assigning an object in F into P_j correspond to the action of coloring a node in V with the j-th color. Then it is clear that $|R_i \cap P_j| \le 1$, for $1 \le i \le n$ and $1 \le j \le m$, if and only no two adjacent vertices are assigned the same color in G. It follows that there exists a partition to allocate all objects in F into m subsets $P_1, P_2, \ldots, P_m$

satisfying $|R_i \cap P_j| \leq 1$ for $1 \leq i \leq n$ and $1 \leq j \leq m$, if and only if graph G can be colored by m colors.

Q.E.D.

Corollary 1. If we are concerned with the worst case performance, Problem 1 is NP-hard.

Proof : This can be proved by showing that Problem 1 is no easier than the problem which has been shown to be NP-complete in the Theorem. In Problem 1, n_{ji} is equal to $|R(q_j) \cap P_i|$ for $1 \leq j \leq n$ and $1 \leq i \leq m$. Therefore, assuming that there exists an allocation method to assign all buckets in F onto m disks and $\max\{T_1, T_2, \ldots, T_n\}$ is minimized, then there exists an allocation method to satisfy $|R_j \cap P_i| \leq 1$ for $1 \leq j \leq n$ and $1 \leq i \leq m$ if and only if $\max\{T_1, T_2, \ldots, T_n\} \leq 1$.

Corollary 2. Problem 2 is also NP-hard.

Proof : Similar to the proof of Corollary 1, Problem 2 can be also shown to be no easier than the problem described in the previous Theorem. Assume that there exists an algorithm to determine the minimum number h of nodes in the network such that k attributes can be partitioned into $P_1, P_2, \ldots, P_h$ disjoint subsets and $|R(q_i \cap P_j| \leq 1$ for $1 \leq i \leq n$ and $1 \leq j \leq h$. Then there exist an allocation to satisfy the problem which proved to be NP-complete in the Theorem if and only if $h \leq m$.

Since an NP-hard problem is computational intractable, we usually take a heuristic approach to find near-optimal solutions (instead optimal solutions) for such a problem. We also note, without proof, that due to the close similarity between Problem 2 and the graph coloring problem, the approximation algorithms proposed by Ghosh which were subsequently modified by Srinivasan and Sankar cannot guarantee that the ratio of their approximation solution to the "true' optimal solution is always less than 2. This is because Garey and Johnson [3] showed that for the graph coloring problem even finding a polynomial time approximation algorithm satisfying A(G) < r.T(G)+d is computational

intractable, where r,d are constants, $r<2$, and A(G) is the number of colors derived by an approximation algorithm A when a graph G is given, and T(G) is the "true" minimum number of colors required.

REFERENCES

[1] Du, H.C. and Sobolewski, J.S., "Disk Allocation for Cartesian Product Files on Multiple-Disk Systems," ACM Trans. Database Systems, Vol. 7, No. 1, March 1982, pp. 82-101.

[2] Du, H.C., "A Heuristic Disk Allocation Method for Binary Cartesian Product Files," Dept. of Computer Science, University of Minnesota (Minneapolis), Tech. Report No. 82-14.

[3] Garey, M.R. and Johnson, D.S., "The Complexity of Near-Optimal Graph Coloring," Journ. ACM, Vol. 23, No. 1, Jan. 1976, pp. 43-49.

[4] Ghosh, S.P., "Distributing a Data Base with Logical Associations on a Computer Network for Parallel Searching," IEEE Trans. Software Eng., Vol. SE-2, No. 2, June 1976, pp. 106-113.

[5] Srinivasan, B. and Sankar, R., "Algorithms to Distribute a Database for Parallel Searching," IEEE Trans. Software Eng., Vol. SE-7, No.1, Jan. 1981, pp. 112.