March 1981 Report. No. STAN-CS-81-847

The Optimal Locking Problem in a Directed Acyclic Graph

by

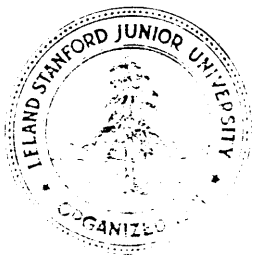
Henry F. Korth

Research sponsored in part by

Air Force Office of Scientific Rcscarch

Department of Computer Science

Stanford University
Stanford, CA 94305



THE OPTIMAL LOCKING PROBLEM IN A DIRECTED ACYCLIC GRAPH

Henry F. Korth†

ABSTRACT

We assume a multiple granularity database locking scheme similar to that of Gray, 'et al. [1975] in which a rooted directed acyclic graph is used to represent the levels of granularity. We prove that even if it is known in advance exactly what database references the transaction will make, it is NP-complete to find the optimal locking strategy for the transaction.

§1 INTRODUCTION

Relatively little is known about the compilation of database transactions. One problem that a transaction compiler might want to solve is the determination of the locks required by the transaction in order to observe the locking protocol of the database system. We shall be using the *unbiased* DAG *protocol* of Korth [1981a] which, like the protocol of Gray, et al. [1975], uses a rooted directed acyclic graph of lockable entities. We show that it is NP-complete to design an optimal locking strategy for a transaction in this model even if all database references by a transaction are known in advance.

This rather negative result is best interpreted not as an indictment of the model, but rather as a statement that it is best to look for good heuristics for transaction compilation than to engage in the almost certainly futile search for a polynomial time algorithm to produce optimal locking strategies.

§2 THE MODEL

We shall give a brief definition of our model here. Gray, ct al. [1975] provides a more detailed and better motivated discussion.

We associate a *lock* with each database *entity*. Entities are the smallest units of data we are willing to lock. There may be many types, or modes of locks. Korth [1981a] describes many classes of lock modes. For the purposes of this paper it suffices that we are given a set of lockmodes $\{\chi_1, \ldots, \chi_n, I\chi_1, \ldots, I\chi_n\}$, where the χ_i are called access modes and the $I\chi_i$ *intention* modes. We shall explain the use of intention modes shortly. Access modes permit their holder some sort of access to the locked entity. Typically, $\{\chi_1, \ldots, \chi_n\}$ is { read, write }.

It is useful to have locks associated with collections of database entities. The *granularity* of an entity is the amount of data associated with that entity. If the amount of data is relatively large we say the granularity is coarse. Likewise, if the amount of data is relatively small, we say the granularity if fine.

It is simple to design transactions that benefit from fine granularity and likewise for coarse granularity. Indeed no one granularity is idea.1 for all cases. System/R offers transactions variable granularity: a transaction may lock using granularities ranging from very coarse (the entire database) to relatively fine (records).

† Work partially supported by AFOSR grant AFOSR-80-0212

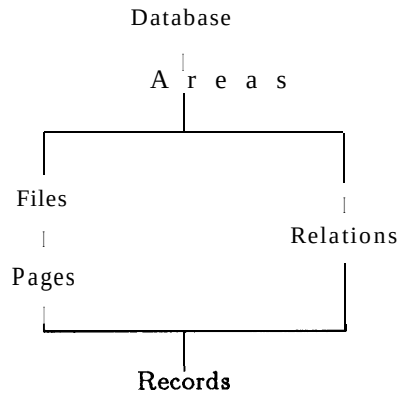


Fig. 2.1. A Lattice of Granularities.

Variable granularity is implemented by means of a rooted directed acyclic graph, **(DAG).** The leaves of the DAG (nodes without any out-edges) represent the finest granularity. Each datum at this finest level of granularity is associated with exactly one leaf. The remaining nodes of the *DAG* (the *interior nodes)* represent the coarser granularities. A node n is associated, for purposes of granularity, with all leaves ℓ such that there is a path in the **DAG** from n to ℓ. Thus, the root node of the DAG represents the coarsest granularity of all: the entire database.†

Example 2.1: (Gray, et al. [1975]). Suppose that we wish to offer the following granularities: records, relations, pages, files, areas, and the entire database. The database consists of a collection of areas. Areas are subdivided into files and relations, but relations may span more than one file. Files contain pages, The smallest unit, records, are contained in pages and in relations. This collection of granularities is shown diagramatically in Fig. 2.1. The DAG used to implement variable granularity has a node r_i for each record; p_i for each page; f_i for each file; ℓ_i for each relation; a_i for each area and R for the root node. The edges of the DAG are:

- (p_i, r_j) for each page p_i and record r_j such that r_j is in p_i.
- (ℓ_i, r_j) for each relation ℓ_i and record r_j such that r_j is in ℓ_i.
- (f_i, p_j) for each file f_i and page p, such that p_j is in file f_i.
- (a,, f_j) for each area a, and file f_j such that f_j is in area a_i.
- (a,, ℓ_j) for each area a_i and relation ℓ_j such that ℓ_j is in area a_i.
- (R, a,) for each area a_i. I

In order to lock data, a transaction may explicitly lock the leaf nodes associated with the desired data. Alternately, the transaction may implicitly lock the requisite leaf nodes to be locked by locking a certain number of parents of the node (Korth [1981b]). The exact number of parents required for implicit locking may depend on the lock mode desired. In this paper, however, we shall require that a majority of parents be locked regardless of the lock mode desired. Implicit locking does *not* apply to the intention modes.

The rules‡ that the *unbiased* DAG *protocol* imposes on transactions are the following:

1. The first lock is taken on the root.

† We emphasize that the DAG represents granularities as opposed to the DAG (or tree) used by Silberschatz and Kedem [1979]. In that paper, the DAG represents a hierarchical data organization.

‡ For simplicity, we have omitted some constraints of the protocol that are not relevant to our result. A precise statement of the protocol appears in Korth [1981a].

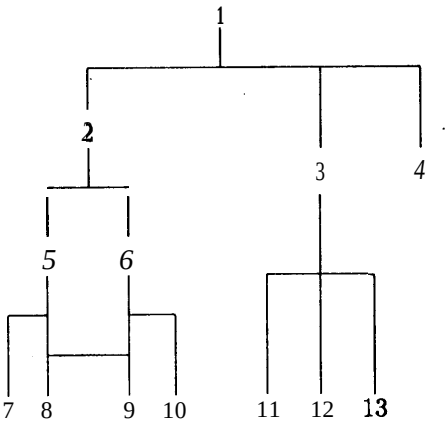


Fig. 2.2. DAG for Example 2.0.1.

lock 1 in I_W.
lock 2 in I_W.
lock 3 in R.
lock 5 in I_W.
lock 6 in I_W.
lock 8 in W.
lock 7 in R.

Fig. 2.3. Sample transaction for Example 2.2

2. Before requesting a lock on node n in mode I_{X_i} or X_i, the transaction must hold a majority of the parents of n in mode I_{X_i}.

We shall not justify these rules here. The relevant correctness proofs appear in Korth [1981b]. The rules do indicate the purpose of intention modes. Mode I_{X_i} serves to warn of the intent to lock a descendant node in mode I_{X_i} or Xi.

Example 2.2: Suppose a transaction wishes to read data nodes 7, 11, 12, and 13 and write node 8 in the DAG of Fig. 2.2. The locking steps of this transaction could be as shown in Fig. 2.3. Note the need to lock both nodes 5 and 6 in I_W mode in order to lock node 8 in W mode. Locking only one of the nodes 5 and 6 would not provide a majority of the parents of node 8. Also note the use of implicit locking by the transaction when it implicitly locked nodes 11, 12, and 13 in R mode by locking node 3 in R mode. I

§3 THE OPTIMAL LOCKING PROBLEM

In general, it is not known at compile time which data the transaction will reference. The best example of this fact are transactions entered interactively by a user at a terminal. It is interesting, however, to ask how well we can optimize transaction lock requests if we know *exactly* what data the transaction will lock.

We shall make two assumptions that make the problem "easier." First, we assume that we know exactly the set C of nodes to be locked in access modes. Second, we asssme that there is only one access mode (X). Our problem is to determine what nodes of the DAG G need to be locked in I_X mode so as to minimize the number of nodes locked. Formally, the *Optimal Locking Problem* is stated as follows:

Given a rooted DAG G, and integer k, and a subset C of the nodes of G, is there a subset L of the nodes of G of cardinality k such that the unbiased DAG protocol can be observed in locking exactly the members of L, and exactly members of C are explicitly locked in an access mode?

We shall show that this problem is NP-complete. The theory of NP-completeness is presented in Aho, Hopcroft, and Ullman [1974] and Garey and Johnson [1979]. Although it is not known whether NP-complete problems have polynomial time algorithms, it is highly unlikely that they do. Before proving that this problem is NP-complete, we shall consider an apparently simpler problem, the majority hitting set problem.

Definition: The *Majority Hitting Set Problem* is stated as follows:
Given a set S, a collection $\{C_1, \ldots, C_n\}$ of n subsets of S, and an integer k, is there a subset T of S such that $|T| = k$ and for all i, T contains a majority (i.e. more that half) of the members of C_i.

Lemma 3.1: The majority hitting set problem is NP-complete.

Proof: In $\mathcal{NP}$: Nondeterministically guess a subset of S of cardinality k and check to see if it *is a* majority hitting set.
NP-hard: We shall make use of the (standard) hitting set problem which is known to be $\mathcal{NP}$-complete†. Let (S, $\{C_1, \ldots, C_n\}$, k) denote an instance of the hitting set problem. We construct an instance (S', $\{C'_1, \ldots, C'_{n+1}\}$, k') of the majority hitting set problem as follows: Let M be the cardinality of the largest C_i, and let $\{a_1, \ldots, a_M\}$ and $\{b_1, \ldots, b_{M-1}\}$ be sets of distinct elements not in S.

$$S' = S \cup \{a_1, \ldots, a_M\} \cup \{b_1, \ldots, b_{M-1}\}$$

$$C'_i = C_i \cup \{a_1, \ldots, a_{|C_i|}\}, \quad i = 1, \ldots, n.$$

$$C'_{n+1} = \{b_1, \ldots, b_{M-1}\} \cup \{a_1, \ldots, a_M\}$$

$$k' = k + M$$

It is clear that the above construction takes only deterministic polynomial time. If there is a hitting set H of cardinality k for S, then there is a majority hitting set H' for S' of cardinality k'. H' is $H \cup \{a_1, \ldots, a_M\}$. It remains to be shown that if there is a majority hitting set H' of cardinality k' for S', then there is a hitting set H of cardinality k for S. If it happens that H' contains all of the a_i, then it is clear that such a hitting set II is $H' - \{a_1, \ldots, a_M\} - \{b_1, \ldots, b_{M-1}\}$. Assume that H' does not contain all of the a_i. In order for the majority hitting set to have a majority of the members of C'_{n+1}, it must have M members of $\{a_1, \ldots, a_M, b_1, \ldots, b_{M-1}\}$. Since the b_i appear nowhere else, any b_i in the majority hitting set may be replaced with some a_j. Therefore, if there is a majority hitting set of cardinality k' then there is a majority hitting set of the same cardinality that contains all of the a_i, and we can find a hitting set of cardinality k for S. ∎

Theorem 3.1: The optimal locking problem is NP-complete.

Proof: In *NP:* Nondeterministically guess a subset of the nodes of G of cardinality k. Verify that exactly the chosen subset locks exactly the members of C in an access mode and that this subset can be acquired in accordance with the protocol.
NP-hard: Let (S, $\{C_1, \ldots, C_n\}$, k') be an instance of the majority hitting set problem. We construct a DAG G of depth 3 as follows:

- There is one node at depth 1 (the root).

†The hitting set problem is stated as follows: Given a set S, a collection $\{C_1, \ldots, C_n\}$ of n subsets of S, and an integer k, is there a subset H of S such that $|H| = k$ and for all i, H contains at least one member of C_i?

- There is one node at depth 2 for each member of S and edges from the root to each depth 2 node.
- There is one depth 3 (leaf) node for each C_i.
- For each depth 2 node a and depth 3 node 6, include edge (a, b) if a is in C_b.

We now construct an instance of the optimal locking problem as follows:

- G is the *DAG* defined above.
- C is the set of all leaves of G.
- $k = k' + n + 1$.

If there is a solution to the optimal locking problem, the solution must lock the root, as well as the n members of C. Therefore, the solution locks exactly k' nodes at depth 2. A depth 2 node a is a parent of a depth 3 node b if and only if a is in C_b. Since for each 6 the solution locks a majority of its parents, the set' of depth 2 nodes locked provides a majority hitting set.

If there is a solution to the majority hitting set problem, then let T be a majority hitting set of cardinality k'. We solve the optimal locking problem by locking the root, the depth 2 nodes corresponding to T, and the leaves. ∎

54 DISCUSSION

The $\mathcal{NP}$-completeness of the optimal locking problem, despite the two assumptions we made to make the problem "easier," is not as disastrous as it initially appears. The *DAG* of the $/P-completeness proof is "bushier" than the type of *DAG* that one might expect of real database management systems. As the sample lattice of granularities of Fig. 2.1 suggests, there is likely to be only a few paths from the root to a given node. In a real system, the choice of paths to a particular node may be aide by the semantics associated with the path. For example, one path may be associated with access to the file containing the datum and another to the relation containing the datum (see Fig. 2.1). In the extreme case that the *DAG* is actually a tree, it is trivial to solve the optimal locking problem.

It is sometimes actually preferable to lock in a sub-optimal manner. It may be cheaper, in terms of overall system performance, to lock an entire file, for example, if many, though not all, records are actually needed. Gray [1981] describes the notion of *lock escalation* in which a transaction that has acquired many locks at a particular granularity trades those locks in for a single lock at a coarser granularity. Use of lock escalation in System/R worked well from a practical point of view.

In summary, although we certainly do not want transactions to acquire excessive numbers of unnecessary locks, we cannot demand that transactions do not ever acquire a larger than optimum number of locks, even under highly idealized assumptions.

REFERENCES:

Aho, A.V., J.E. Hopcroft, and J.D. Ullman [1974], *The Design and Analysis of Computer Algorithms,* Addison- Wesley.

Garey, M.R., and D.S. Johnson [1979], *Computers and Intractibility: A Guide to the Theory of $\mathcal{NP}$-Completeness,* W.H. Freeman and Company, San Francisco, CA.

Gray, J.N. [1978], "Notes on Database Operating Systems," RJ 2188, IBM Research Laboratory, San Jose, CA, also in R. Bayer, R.M. Graham, and G. Seegmuller, eds., "Operating Systems - An Advanced Course," *Lecture* Notes *in* Computer *Science,* Springer-Verlag, Berlin.

Gray, J.N. [1980], "A Transaction Model," in "Automata, Lanugages, and Programming, Seventh Colloquium, Nuordwijkerhout," *Lecture* Notes *in Computer Science,* 85, Springer-Verlag, Berlin.

Gray, J.N. [1981], "Experience With the System/R Lock Manager," Oral Presentation, 5 Berkeley Conference on Distributed Data Management and Compter Networks.

Gray, J.N., R.A. Lorie, G.R. Putzolu, and I.L.Traiger [1975], "Granularity of Locks and Degrees of Consistency in a Shared Data Base" RJ 1654, IBM Research Laboratory, San Jose, CA.

Korth, H.F. [1981a], "Locking Primitives In A Database System,"submitted for publication.

Korth, H.F. [1981b], " Locking Protocols: General Lock Classes and Deadlock Freedom," Ph.D. dissertation, Princeton University, Princeton, NJ.

Silberschatz, A. and Z. Kedem [1979], "Consistency in Hierarchical Database Systems," *J. ACM* 27:1, pp. 72-80.

Ullman, J.D. [1980], *Principles of Database Systems,* Computer Science Press.