MERGING PARTITIONED DATABASES

David D. Wright
Dale Skeen

TR 83-547
April 1983
(Revised August 1983)

Department of Computer Science
Cornell University
Ithaca, New York 14853

*This work is supported in part by NSF grant MCS-8103605.

Merging Partitioned Databases*

David D. Wright
Dale Skeen

Department of Computer Science
Cornell University
Ithaca, New York 14853

April 10, 1983

ABSTRACT

Partitioning of a distributed data base requires either that update activity be restricted or that a strategy for conflict resolution and partition merging be used once communication is restored. The graph-theoretic approach used by Davidson follows the latter approach and can be used to show that finding an optimum solution to the general problem is NP-complete. We give several methods of reducing the size of the graphs involved. Two open subproblems are shown to be NP-complete, while an extension of a known polynomial-time subproblem is given. Simulation results are used to study both the amount of compression achieved by the graph reduction techniques and their effects on heuristics for the problem. In addition, some modifications are made to existing heuristics to improve their performance. A probabilistic model is presented and compared with the simulations.

*This work is supported in part by NSF Grant MCS-8103605.

1. Assumptions and Definitions

A distributed database (DDB) consists of a set of *data-items* $\{d_1, ..., d_M\}$ and a set of *sites* $\{s_1, ..., s_N\}$ that are connected by bidirectional communication links. We assume that during normal operation any site can send a message to any other site. The values of data-items are manipulated by *transactions*, which have the property that their actions are atomic from the point of view of any other transaction. Every transaction T has associated with it two sets, READSET(T) and WRITESET(T), which are, respectively, the sets of items read and written by T. We assume that WRITESET(T) $\subseteq$ READSET(T), and that data are fully replicated, i.e. that there is a copy of every data-item at every site.

A *partition* of the DDB is a maximal subset of communicating sites. Site or communication link failures may separate the DDB into more than one partition, whereas site or link recoveries may require that partitions be merged. The system is said to be *partitioned* at any time that it is composed of more than one partition. We assume that within a partition the transaction activity corresponds to some serial schedule, and that this schedule can be derived by any member of the partition (using the history described below).

1.1. Description of the Problem

Strategies to allow a partitioned DDB to continue functioning generally fall into one of two categories. Most previous methods have guaranteed that all conflicts are avoided. This category includes voting schemes [Giff79, Thom78], token passing, and so on. By nature, these policies are extremely conservative. A problem with such schemes is that failures may occur in such a way that no partition can perform updates; for example, in a voting scheme, no partition may have a majority. In contrast, the second category of algorithms is probabilistic and allows each partition to perform updates that might conflict with updates in another partition. Such strategies are called *optimistic* by Davidson [Davi82] because they assume a small number of conflicts. When communication between two or more partitions is restored, the partitions must compare their actions during partitioning, detect conflicts, and then back out (undo) transactions until no conflicts remain. To simplify the discussion, we assume that only two partitions are to be reconnected at a time. However, the method described generalizes to any number of partitions.

The approach used herein requires that whenever the DDB is partitioned, each partition P_i must maintain a history of the READSET and WRITESET of all transactions T_{ij} that have committed (successfully completed) in that partition since it was formed, along with the order in which the transactions committed. When two partitions are reconnected, each partition derives a serial schedule H_i for its partition using the history it has maintained. The serial schedules are then used to construct a precedence graph.

Definition 1.1 [Davi82]. Let $H_1 = T_{11}, T_{12}, \ldots, T_{1n}$ and $H_2 = T_{21}, T_{22}, \ldots, T_{2m}$ be serial schedules from partitions P_1 and P_2 respectively. The *precedence graph* $G(H_1, H_2) = (V, E)$ is the digraph defined by:

(1) $V = \{T_{11}, \ldots, T_{1n}\} \cup \{T_{21}, \ldots, T_{2m}\}$.

(2) E = {Dependency Edges} ∪ {Precedence Edges} ∪ {Interference Edges}, where

(a) Dependency Edges[1] –there is a dependency edge $T_{si} \rightarrow T_{sk}$ iff $i<k$ and there is a data-item d such that $d \in WRITESET(T_{si}) \cap READSET(T_{sk})$ and $(\forall j\colon i<j<k\colon d \notin WRITESET(T_{sj}))$.

(b) Precedence Edges–there is a precedence edge $T_{si} \rightarrow T_{sk}$ iff $i<k$, there is no dependency edge $T_{si} \rightarrow T_{sk}$, and there is a data-item d such that $d \in READSET(T_{si}) \cap WRITESET(T_{sk})$ and $(\forall j\colon i<j<k\colon d \notin WRITESET(T_{sj}))$.

(c) Interference Edges–there is an interference edge $T_{1i} \rightarrow T_{2j}$ iff there is a data-item d such that $d \in READSET(T_{1i}) \cap WRITESET(T_{2j})$. (Similarly for $T_{2i} \rightarrow T_{1j}$.)

For each vertex $v \in V$, the *dependency set* of v, $dep(v)$, is the set of all vertices v' in V s.t. there is a path of dependency edges from v *to* v'. For example, in Fig. 1.1, the dependency set of a is $\{a,d\}$, and $dep(c) = \{a,b,c,d\}$. If v is backed out, then $dep(v)$ must be backed out, since the values read by the transactions in $dep(v)$ depend on the values written by v. We will use the convention that if S is a set of vertices, $dep(S)$ is the union of the dependency sets of the members of S.

---

[1] Called *ripple edges* in [Davi82].

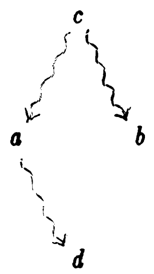


Fig. 1.1. Transactions connected by dependency edges.

[Davi82] contains a proof that $G(H_1,H_2)$ is acyclic iff the combined set of transactions is serializable. Thus, if G contains cycles and we wish to find a serializable subset of $H_1 \cup H_2$, some transactions (and their dependency sets) must be backed out until the graph is acyclic. A set S of vertices with the property that its removal leaves the graph acyclic is called a *feedback vertex set.*

We can now state the *Partition Merging Problem* (PM): given a precedence graph $G=(V,E)$ and an integer $K>0$, is there a set $S \subseteq V$ such that $dep(S)$ is a feedback vertex set of G and $|dep(S)| \leq K$? Such a set S is called a *transaction backout set.* The quantity $|dep(S)|$ is called the *weight* of the set S.

2. Some Properties of Precedence Graphs

This section contains some important lemmas about the properties of precedence graphs. The point of these lemmas is that certain edges may be added or deleted without affecting the solution to an instance of PM.

Lemma 2.1: Suppose $G=(V,E)$ is a precedence graph, $a,b,c \in V$, $(a,b) \in E$ is any type of edge, $(b,c) \in E$ is a dependency edge, and $(a,c) \notin E$. Let $G' = (V, E \cup \{(a,c)\})$, where (a,c) is of the same type as (a,b). Then $S \subseteq V$ is a backout set of G $\Leftrightarrow$ S is a backout set of G'.

Proof: $(\Leftarrow)$ Trivial, since G is a subgraph of G'. $(\Rightarrow)$ Suppose S is a backout set of G. Any cycle that includes (a,c) has a corresponding cycle in which (a,c) is replaced by (a,b) and (b,c). All of the latter are broken by S. This could be done by removing a, b, c, or some other nodes on the cycles involving (a,b) and (b,c). However, *any* of those removals will also break all cycles involving (a,c).

Thus, S must be a backout set of G'. □

One useful consequence of this lemma is that an interference edge need only be drawn from a transaction T to the *first* transaction in the other partition that writes any item read by T. This simplifies constructing precedence graphs.

Employing the above lemma will never increase the number of strongly-connected components in the graph. In particular, individual nodes will never become disconnected. The next lemmas illustrate that in some cases, edges can be removed in such a way that the number of components may increase.

Lemma 2.2: Let $G=(V,E)$ be a precedence graph, where $(a,b)\in E$ and $(b,a)\in E$ are interference edges, $c\in dep(a)$, $d\in dep(b)$, $(c,d)\in E$, and $(c,d)\neq(a,b)$. Then $S\subseteq V$ is a backout set of $G \Leftrightarrow S$ is a backout set of $G'=(V,E-\{(c,d)\})$.

Proof: ($\Rightarrow$) Trivial, since G' is a subgraph of G. ($\Leftarrow$) Suppose S is a backout set of G' but not of G. Any cycle not broken in G must include (c,d), since all other edges of G are in G'. But the 2-cycle[2] involving a and b is in G' and can only be broken by deleting a or b. Deleting either of these will result in removing (c,d), so that edge cannot be involved in a cycle in G, a contradiction. Hence, S must be a backout set of G. □

Lemma 2.3: Let $G=(V,E)$ be a precedence graph, where $(a,b)\in E$ and $(c,b)\in E$ are interference edges, $c\in dep(a)$, $c\neq b$. Suppose that all cycles involving (c,b) pass through a. Then $S\subseteq V$ is a backout set of $G \Leftrightarrow S$ is a backout set of $G'=(V,E-\{(c,b)\})$.

Proof: ($\Rightarrow$) Trivial, since G' is a subgraph of G. ($\Leftarrow$) Suppose that S is a backout set of G'. Note that every cycle C involving (c,b) has a corresponding cycle in which the section of C from a to c and (c,b) is replaced by (a,b). This latter cycle is broken by S. This could be done by removing a, b, or some other transaction on the path from b to a; however, any of these removals will also break C. □

One easy method of constructing an equivalent precedence graph is to remove all edges (and thus vertices) that do not lie on a cycle. This can be done by finding the strongly connected components of the graph and deleting all edges that do not connect members of the same component. This requires node weights (if needed) to be stored at the nodes, as there would not necessarily be any way to compute the

[2] A *k-cycle* is a cycle of length *k*.

weights from the reduced graph.

3. Computational Complexity of the Partition Merging Problem

This section contains several theoretical results about the partition merging problem. The first is

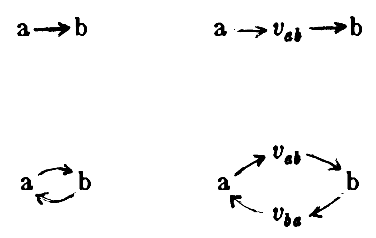
Theorem 3.1: The partition merging problem is NP-complete.[3]

Proof: We will show that an instance of the feedback vertex set problem (does an arbitrary digraph G have a feedback vertex set of size $\leq K$?) is polynomially transformable to an instance of the partition merging problem. (For a proof that the feedback vertex set problem is NP-complete, see e.g. [AHU74].) Let $G=(V,E)$ be an arbitrary directed graph. We will show how to construct in polynomial time a graph $G'=(V',E')$ such that G' is a precedence graph with a transaction backout set of weight $\leq K$ iff G has a feedback vertex set of size $\leq K$. It is obvious that we can determine in polynomial time if some set $S \subseteq V$ is a transaction backout set of a precedence graph.

Define G' as follows: let $V' = V \cup \{v_{ab}: (a,b) \in E\}$, and $E' = \{(a,v_{ab}), (v_{ab},b): (a,b) \in E\}$. That is, replace each edge $(a,b) \in E$ by two edges, (a,v_{ab}) and (v_{ab},b). (See Fig. 3.1.)

Let $T_1=V$ and $T_2=(V'-V)$, with H_1 and H_2 arbitrary orderings of T_1 and T_2. Let E' consist solely of interference edges. Then G' is a precedence graph, and $(\forall v \in V': dep(v)=\{v\})$.

Note that the cycles of G and G' are in a 1-to-1 correspondence. Clearly, any feedback vertex set S of G is a feedback vertex set (and thus a transaction backout set) of G', and since each $v \in V'$ has a



in G in G'

Fig. 3.1. Mapping G to G'.

[3] [Davi82] contains a similar, independently developed, proof.

dependency set consisting only of itself, $|S| \leq K$ iff weight$(S) \leq K$.

Now suppose that S is a transaction backout set of G'. Then there is a transaction backout set that is no larger than S and contains only members of V. To see this, notice that if there were some $v \in S$ with $v = v_{ab}$, we could replace it in S by b, since any cycle that includes v_{ab} must include b. Call S' the set derived from S by doing all such replacements. Since $(\forall v \in V': dep(v) = \{v\})$, $|S'| \leq K$ and S' is a feedback vertex set of G. □

This proof holds only if the number of items in the database is unbounded, since the number of items needed is of the order of the number of edges in the graph G. If the number of items is a constant, the problem can be solved in polynomial time [Wrig]; however, it seems likely that in those environments for which the optimistic protocol is suited, the number of data items will be much larger than the number of transactions.

In a real database system, one partition might be "privileged" in that its transactions would never be backed out. This restriction does not help, however; since the construction used in the previous proof only removes nodes from one partition, we can conclude

Corollary 3.1: The partition merging problem with one partition ineligible for backout is NP-complete.

Even if a problem is NP-complete, it may have subproblems that can be solved in polynomial time. An example is:

Theorem 3.2 [Davi82]: Let $G=(V,E)$ be a precedence graph for two partitions such that all edges are either dependency edges or are interference edges that lie on 2-cycles. Then a solution to PM can be found in polynomial time. □

Davidson's proof actually allows a more general conclusion. Suppose $P \subseteq E$ be the set of precedence edges in E. Then any $S \subseteq E$ is a backout set for G iff it is a backout set for $G' = (V, E-P)$. To see this, suppose S is a backout set of G'. Then it must break all 2-cycles in G' (all of which are also in G). The only way to do this is to remove one or both vertices lying on each 2-cycle. This has the effect of removing *all* interference edges in G' (hence G). Since a graph with no interference edges can have no cycles, S is a backout set of G.

The above theorem is particularly useful since, for most environments, we would expect that very few cycles will be left after all 2-cycles are broken.

Unfortunately, Theorem 3.2 does not extend to more than 2 partitions. We will show this using a restricted form of the problem "does G have a vertex cover[4] of size $\leq K$?" (VC), which is NP-complete, even for cubic planar graphs[5] [GaJo79]. We will first show that a restricted version of VC on cubic planar graphs is NP-complete. Next, we will show how to convert an instance of this problem to an instance of PM that has only three partitions and all interference edges lying on 2-cycles.

Lemma 3.1: The vertex cover problem is NP-complete for cubic planar graphs that do not have K_4 as a component.

Proof: Let G be a cubic planar graph. We can easily find the K_4 components of G in polynomial time. (They are simply those components containing exactly 4 vertices.) Assume that there are m such components, and let G' be the subgraph of G formed by deleting all those components. Since K_4 has a vertex cover consisting of any 3 of its vertices, G' has a vertex cover of size K iff G has a cover of size $K+3m$. □

Lemma 3.2: Every cubic planar graph $G=(V,E)$ that does not have K_4 as a component can be 3-colored in polynomial time.

Proof: see [Lova75].

Theorem 3.3: PM is NP-complete for precedence graphs of 3 or more partitions even if all interference edges lie on 2-cycles.

Proof: Let $G=(V,E)$ be a cubic planar graph that does not contain K_4. We will construct in polynomial time a precedence graph $G'=(V,E')$ for 3 partitions with the property that G has a vertex cover of size $\leq K$ iff G' has a backout set of size $\leq K$. Define G' as follows: replace each edge $\{a,b\} \in E$ in G with two directed edges (a,b) and (b,a) in E'. Let each edge in E' be an interference edge. Construct a 3-coloring for G, and use the colors as names for the partitions in G'. Then there will be no interference edges connecting two members of the same partition. Suppose $S \subseteq V$ is a vertex cover

[4] A subset S of V such that every edge in E is incident on some member of S.

[5] A graph is *cubic* if all of its vertices have degree 3.

of G but not a backout set of G'. But every edge in G (hence G') is incident on a member of S and hence no cycle can exist. Contrariwise, suppose $S \subseteq V$ is a backout set of G'. In particular, this means that all 2-cycles of G' are broken by S. But every edge in G' is on a 2-cycle (corresponding to an edge in G) and hence S must be a vertex cover of G. □

Finally, we note in the following lemma that Theorem 3.2 also does not extend to graphs with longer cycles.

Lemma 3.3: PM on graphs with no 2-cycles and for which every interference edge lies on a 3-cycle is NP-complete.

Proof: We reduce VC to this restricted form of PM. Let $G=(V,E)$ be an undirected graph. We will construct a precedence graph $G'=(V',E')$ such that G' has a transaction backout set of size $\leq K$ iff G has a vertex cover of size $\leq K$. First, number the vertices of V. For each edge $\{a,b\} \in V$, where $a<b$ in the numbering, add a precedence edge (a,b) to E'. In addition, add vertices a, b, and v_{ba} to V', and add interference edges (b,v_{ba}) and (v_{ba},a) to E'. G' is then a precedence graph, with the vertices from V in one partition, and the vertices v_{ba} in the other. Any vertex cover of G will be a transaction backout set of G', and any transaction backout set of G' can be converted to a vertex cover of G by replacing any v_{ba} vertices with b. □

3.1. Approximating PM

For some NP-complete problems, efficient polynomial-time heuristics with good worst-case error bounds have been found. (Examples can be found in [GaJo79].) Error is measured as the relative size of the solution found by the heuristic to the actual minimum. For PM, straightforward methods such as iteratively backing out the transaction of lowest weight until the graph is acyclic have worst-case errors of $O(n)$ for a graph of n nodes. The appendix contains an example of a graph on which most obvious heuristics we have tried fail with $O(n)$ error. In this section, we first show how to reduce an instance of PM to FVS. We will then present an approximation for FVS with a worst-case error of $O(n/log(n))$, which is the best bound known.

Lemma 3.4: Let I be an instance of PM on graph G with size limit K. Then we can convert I to an instance I' of FVS such that I has a solution of size $\leq K$ iff I' does.

Proof: We convert I to I' by adding edges as follows. For each node v such that there are interference edges of the form (v,x) in G, add an edge (v',x) for each $v' \in dep(v)$. In addition, for each edge of the form (x,v), add an edge (x,v') for each $v' \in dep(v)$. These changes have the effect of involving each member of $dep(v)$ with a cycle corresponding to any cycle involving v. Thus, if v is chosen as a non-useless part of a solution to I, it will be necessary to also choose all members of $dep(v)$. □

Now consider an arbitrary digraph G. By constructing the strongly-connected components of G, we can determine in polynomial time if G contains a cycle. If it does not, then finding a minimum feedback vertex sèt (fvs) is trivial (the empty set is such a set). Suppose G does contain one or more cycles. Divide the vertices of G up into $log(n)$ subsets of size $n/log(n)$. Since $2^{log(n)}=n$, in polynomial time we can consider all possible combinations of these subsets and find the minimum combination that is a fvs. Suppose a minimum fvs M of G contains k nodes, where $k \leq log(n)$. The nodes in M can lie in at most k different subsets, so the maximum error from using the subsets is $n/log(n)$. If $k \geq log(n)$, then simply picking $n-1$ vertices will give a fvs of error $< n/log(n)$. We have thus proved

Theorem 3.5: Let I be an instance of FVS on an n-vertex graph G and let $OPT(I)$ be the size of the smallest feedback vertex set for G. There is an approximation that finds a feedback vertex set S of G such that $|S|/OPT(I) \leq O(n/log(n))$. □

4. Simulation of Backout Strategies

The fact that a problem is NP-complete does not mean that instances of it difficult to solve, only that they may be difficult to solve optimally. For the optimistic protocol to be of practical importance, however, efficient heuristics that usually give good results must be found. Given that the best heuristic known has a worst-case error bound of $O(n/log(n))$, which is too large to be of any practical significance, the decision was made to restrict attention to fast (usually linear) heuristics with $O(n)$ possible error to see if their performance was adequate.

To gain understanding of heuristic solutions, and to compare the relative performance of different solutions, computer simulation of random transactions in a 2-partition DDB was used. (This approach was also taken in [Davi82], which contains independent results of simulating several strategies on 2-partition DDBs. The results below reflect a more extensive set of observations.) Details on the simulation are contained in the Appendix.

4.1. Using Reduced Graphs

One important simulation goal was to discover what effect the use of reduced graphs would have on the backout rates of various heuristics. It seemed possible that the graph reductions might either cause improvement (by eliminating spurious nodes from consideration) or degradation (by eliminating significant nodes from consideration). In practice, neither of these occurred: results with extensive reduction (making full use of the graph reduction lemmas) and simple reduction (strongly-connected component construction only) had no significant differences between them.

Although reduction does not cause major improvements in backout rates, it has other merits, as the running time of some heuristics can be significantly improved by applying them to reduced graphs. Davidson's method for optimally breaking 2-cycles has time complexity $O(n^3)$ for an n-vertex graph. The advantages of reduction here are obvious.[6] In addition, construction of precedence graphs was simplified by Lemma 2.1, since interference edges need only be drawn to the first node that writes a given item.

A number of simulation runs were made to estimate the amount of compression that reduction can give. The appendix contains a graph illustrating the degree of compression as a function of the backout rate. About 3/4 of the compression can be attributed to strongly-connected component construction, with the rest due to redundant interference edge elimination.

Preliminary simulation results suggested that some heuristics would perform better if they were restricted to backing out transactions in only one partition. This proved to be the case for several heuristics; the appendix contains a graph illustrating the improvement for the strategy that removes the node of

[6] Some restrictions on reduction would be necessary in this case.

minimum weight until the graph is acyclic. Since restriction is not symmetric (restriction to one partition need not give the same results as restriction to the other), the problem is then to pick the better partition. Graph reduction provides a partial solution: even if the size of each partition is initially the same, in general the sizes differ after reduction. The smaller partition can then be used by heuristics that only consider one partition.

The amount of improvement given by this modification over the original strategies varied depending on the backout rate. When the rate was low ($< 5\%$), differences were insignificant. However, as the rate rose, the modified strategies began to gain.[7] This is particularly significant because the improvement is greater than the differences between individual strategies, which were generally less than 5%.

Unfortunately, relative partition size is only a guide. If the partitions were approximately the same size after reduction, applying the heuristics to the larger partition sometimes gave better results. In such cases, trying the heuristics on both partitions and using the better result improved performance by about 10% in high-backout situations.

4.2. Using Optimistic Commit

For optimistic commit to be worthwhile in a DDB, the costs of the protocol must be outweighed by the costs of delaying transactions. We can express this relationship for n transactions as follows:

$D\frac{n}{2} > A(n) + (B + C)b(n)n$, where

$A(n)$ is the cost of running the backout selection algorithm

B is the average cost of backing out a transaction (may include a penalty cost)

C is the average cost of re-executing a transaction

D is the average cost of delaying a transaction

$b(n)$ is the fraction of transactions backed out

The $\frac{n}{2}$ term on the left side of the inequality reflects that an ordinary DDB, when partitioned, would typically be able to run about half of the transactions submitted (using voting or some other

[7] In this and most other cases, the number of simulation samples was 150 to 200, which was sufficient to make the 95% confidence intervals on the backout rates less than one percentage point.

method). In the strategies used in our simulations, $A(n)=An$. In this case, we can rearrange this equation to $\frac{D-2A}{2(B+C)} > b(n)$. For values of n satisfying this inequality, it will be advantageous to use optimistic commit. The parameters involved (except D) should be fairly easy to estimate for a given database system.

5. A Probabilistic Model

In this section, we develop a probabilistic model that can be used to estimate the backout rate of a set of transactions as a function of the parameters used by the computer simulation. These parameters are:

(1) *N1*, *N2* number of transactions in partition 1 and 2, respectively

(2) M data-items in the database

(3) I average no. of data-items referenced by a transaction

(4) RO average percentage of readonly transactions

(5) U average percentage of data-items updated in each non-readonly transaction

We consider two types of conflict -- 2-cycles, and direct dependency effects of 2-cycles -- and estimate the number of nodes removed due to both types.[8]

First, we estimate NTC, the number of nodes in one partition that are involved in write-write 2-cycles, which should be the most common kind. (We will consider *P1* for purposes of exposition. In addition, we will use RO and U as fractions rather than percentages in order to simplify the formulas.) Readonly transactions cannot be involved in 2-cycles, so we will consider only read-write transactions, of which there are $N1(1-RO)=NRW$. The average number of items updated by such a transaction is $I*U=ANU$. Let TNU be the total number of items updated in *P2* (we will estimate this in a moment). Then for an update transaction T in *P1*, the probability that a given item T writes is *not* written in *P2* is $\left(\frac{M-TNU}{M}\right)$, and the probability that *no* item T writes is written in *P2* is $\left(\frac{M-TNU}{M}\right)^{ANU}$. Thus, the probability that *some* item written by T is also written in *P2* (i.e. that T is in a 2-cycle) is

[8] [Davi82] considers the same effects, but using a much more restricted model. The model of this section degenerates to

$1-\left(\frac{M-TNU}{M}\right)^{ANU}=PTC$. (This is not exact, since individual transactions do not use selection with replacement; however, it is sufficient as long as $M \gg TNU$ and ANU is small. If either of these does not hold, the optimistic protocol is not likely to perform well in any case. For the sake of completeness, we note that the exact formula for PTC is $1-\frac{\binom{M-TNU}{ANU}}{\binom{M}{ANU}}$. Similar corrections can be made to the other formulas in this section; however, the error is small and the exact values are considerably more difficult to compute, particularly if TNU or ANU are not integers.) This model considers only write-write conflicts; happily, the chances of a 2-cycle caused strictly by simultaneous read-write conflicts are very small and can safely be disregarded ([Davi82] contains similar conclusions). Once PTC is known, the expected number of transactions in $P1$ involved in 2-cycles is simply $PTC*NRW=NTC$.

We now consider estimating TNU. First, we note that the total number of writes made by transactions in $P2$ is $N2*I*(1-RO)*U=NUP$. For a given data-item, the chance that it will not be updated by a particular write is $1-\frac{1}{M}$, so the chance that it will not be updated at all is $\left(1-\frac{1}{M}\right)^{NUP}$. The chance that it will be updated is then $1-\left(1-\frac{1}{M}\right)^{NUP}$, and the expected number of items updated in $P2$ is $M\left(1-\left(1-\frac{1}{M}\right)^{NUP}\right)=TNU$. (Note that if $M \gg NUP$, we can approximate TNU with NUP with no significant loss of accuracy.)

Finally, we estimate NRD, the number of nodes backed out because they are dependency children of nodes in NTC. The relevant nodes are those that are not involved in 2-cycles, but have dependency parents that are. First, let XTC be the number of nodes not involved in 2-cycles; this is just $N1-NTC$. Thus, the probability that a node in $P1$ is not involved in a 2-cycle is $\frac{XTC}{N1}$. Only read/write transactions can have dependency descendants or be involved in 2-cycles, and the expected number of read/write transactions preceding T_k, where T_k is the kth node in the serial schedule for $P1$, is $(k-1)(1-RO)$. The probability that a given predecessor of T_k is both in a 2-cycle and a dependency parent of T_k is

Davidson's model when $RO=0$.

$\frac{NTC}{N1}\left(1-\left(\frac{M-I}{M}\right)^{ANU}\right)$, so the probability that a predecessor is either not in a 2-cycle or not a dependency parent of T_k is $1-\frac{NTC}{N1}\left(1-\left(\frac{M-I}{M}\right)^{ANU}\right)$, and the probability that *all* predecessors are either not in a 2-cycle or not dependency parents of T_k is $\left(1-\frac{NTC}{N1}\left(1-\left(\frac{M-I}{M}\right)^{ANU}\right)\right)^{(k-1)(1-RO)}$. From this, we see that the probability that at least one of the predecessors of T_k is in a 2-cycle and is a dependency parent is

$$(*)\ 1-\left(1-\frac{NTC}{N1}\left(1-\left(\frac{M-I}{M}\right)^{ANU}\right)\right)^{(k-1)(1-RO)}.$$

In order to find the expected number of nodes removed strictly due to dependency effects, we must sum (*) from $k=1$ to $k=N1$ and multiply by the expected fraction of non-2-cycle nodes ($XTC/N1$). First, we define $\alpha=\left(1-\frac{NTC}{N1}\left(1-\left(\frac{M-I}{M}\right)^{ANU}\right)\right)^{(1-RO)}$. Then the expected number of nodes removed due to dependency effects is $\frac{XTC}{N1}\sum_{k=1}^{N1}1-\alpha^{k-1}$, or $\frac{XTC}{N1}\left(N1-\sum_{k=1}^{N1}\alpha^{k-1}\right)$. As long as the problem is non-trivial (i.e. $U > 0$, $RO < 100\%$, etc.) $0<\alpha<1$, and we can use the identity $\sum_{k=1}^{\infty}\alpha^{k-1}=\frac{1}{1-\alpha}$ to rewrite the summation as $\left(\frac{XTC}{N1}\right)\left(N1-\frac{1-\alpha^{N1}}{1-\alpha}\right)=NRD$, making the total number of nodes backed out $NTC+NRD$.

5.1. Adding More Detail

We now consider estimating "second-level" dependency effects; that is, we wish to estimate the number of nodes removed strictly due to having dependency parents in NRD. First, we note that there are $XTCRD=N1-(NTC+NRD)$ nodes that could be backed out in this fashion. Suppose T_k is the kth node in the serial schedule for $P1$; the probability that T_k is not enumerated in NTC or NRD is $\frac{XTCRD}{N1}$. The probability that a specific predecessor of T_k is read/write transaction in NRD is $NRDRW=\frac{NRD}{N1}\frac{NRW-NTC}{N1-NTC}$. The probability that a read/write predecessor is a dependency ancestor of T_k is $1-\left(\frac{M-I}{M}\right)^{ANU}$, so the probability that a predecessor is both in NRD and a dependency ancestor of T_k is $NRDRW\left(1-\left(\frac{M-I}{M}\right)^{ANU}\right)$. The probability that all predecessors of T_k are not in NRD or not dependency ancestors is then $1-\left(1-NRDRW\left(1-\left(\frac{M-I}{M}\right)^{ANU}\right)\right)^{(k-1)(1-RO)}$. If we let $\beta=\left(1-NRDRW\left(1-\left(\frac{M-I}{M}\right)^{ANU}\right)\right)^{(1-RO)}$ then by a similar process to that used in the last section to compute NRD, the number of second-level dependency descendants, $NRD_2 = \frac{XTCRD}{N1}\left(N1-\frac{1-\beta^{N1}}{1-\beta}\right)$.

This value, although nonzero, is small and can be omitted without significant loss of accuracy (it is typically a few percent of $NTC+NRD$).

5.2. Partially Replicated Databases

Not all DDBs have fully-replicated data. We now consider estimating the backout rates in partitioned databases where each partition has access to only part of the data. Suppose *P1* has access to M_1 items, *P2* has access to M_2 items, and between *P1* and *P2* they have access to the entire database. In addition, let $M_{overlap}$ be the number of items in common (this is just M_1+M_2-M). We can modify our formulas to reflect these changes as follows:

$$PTC = 1-\left(\frac{M_1-\frac{TNU}{M_2}M_{overlap}}{M_1}\right)^{ANU}, \quad TNU = M_2\left(1-\left(1-\frac{1}{M_2}\right)^{NUP}\right)$$

and in α, replace M by M_1. (The scale factor on TNU in the revised formula for PTC reflects the fraction of M_2 visible in *P1*.)

5.3. Non-overlapping Writesets

Suppose each partition has read access to the entire database, but each item can only be written in one partition. Such a strategy will eliminate write-write 2-cycles and thus should reduce backout rates. We will estimate the probability that two read/write transactions are involved in a 2-cycle. This value for PTC can then be used in the formulas of earlier sections to compute numbers of transactions backed out. Let T_1 and T_2 be read/write transactions from partitions *P1* and *P2* respectively, and define

$$WR = P(WRITESET(T_1) \cap READSET(T_2)=\phi) = \left(\frac{M-ANU}{M}\right)^{I},$$

$$RW = P(READSET(T_1) \cap WRITESET(T_2) = \phi) = \left(\frac{M-I}{M}\right)^{ANU}.$$

Then $P(T_1$ and T_2 are in a 2-cycle$) = (1-WR)(1-RW)$, and the probability that T_1 is in a 2-cycle with some transaction in *P2* (i.e. PTC) is $1-(WR + RW - WR * RW)^{(1-RO)N2}$.

5.4. Comparison of Simulation with the Probabilistic Model

The probabilistic model was compared with a wide variety of simulation runs in order to determine the agreement between the two. Agreement was generally excellent. One flaw is that the estimate of PTC is less accurate when TNU is close to M. However, the simulation is affected as least as much by differences in such things as the variance of I.

The estimates computed for conflicts when writesets do not overlap was too low and not as accurate as the other examples. This seems to be due to PTC being too small, since the value arrived at is based on average transaction size and does not allow for large read/write transactions being present in the other partition. However, the accuracy is reasonable and does illustrate the system behavior.

6. Discussion

The significant results of this paper are

(1) New techniques allowing significant reduction in the size of precedence graphs.

(2) New complexity results, particularly the case of multiple partitions in which all interference edges lie on 2-cycles.

(3) Improvements in heuristics and analysis of the impact that reduction has on heuristic performance.

The graph reduction results are important for several reasons. First, they simplify construction of the precedence graph, since interference edges need only run to the first node that writes a given item, not to all of them. This reduces the memory needed to construct the graph, since the graph-construction program needs only a table of "first writes," not a list of all writes. Second, they make the graph smaller. Third, their use does not degrade the performance of the heuristics studied in this paper. Fourth, and perhaps most important, restricting heuristics to one partition only and using the smaller partition after reduction can improve the backout rate of the heuristics studied.

7. Acknowledgements

We would like to thank M. Krentel, J. Gilbert, and B. Aspvall for their helpful advice, and K. Birman for reading an earlier version of this paper. Several useful suggestions were also made by some anonymous referees.

8. References

[AHU74] Aho, A.V., J.E. Hopcroft, and J.D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974.

[Davi82] Davidson, S., "An Optimistic Protocol for Partitioned Distributed Database Systems", PhD Thesis, Princeton University, Dept. of EECS, 1982.

[Giff79] Gifford, D.K., "Weighted Voting for Replicated Data", 7th Symposium on Operating System Principles, Dec. 1979, pp.150-162.

[GaJo79] Garey, M.R., and D.S. Johnson, *Computers and Intractability*, W.H. Freeman & Co., 1979.

[Lova75] Lovasz, L., "Three Short Proofs in Graph Theory", J. Combinatorial Theory B, 19, 111-113.

[Thom78] Thomas, R.H., "A Solution to the Concurrency Control Problem for Multiple Copy Databases", IEEE Compcon, Spring 1978, pp.56-62.

[Wrig] Wright, D.D. Ph.D. thesis, in preparation.

9. Appendix

This section discusses some details of the simulation and provides sample graphs illustrating the sensitivity of the simulation output to changes in the parameters. To recapitulate, the parameters were:

(1) $N1$, $N2$ number of transactions in partition 1 and 2, respectively

(2) M data-items in the database

(3) I average no. of data-items referenced by a transaction

(4) RO average percentage of readonly transactions

(5) U average percentage of data-items updated in each non-readonly transaction

The simulations used pseudo-randomly generated transactions with the parameters above. Data-item references were uniformly distributed over the database in the simulation.[9] Update-set sizes had a binomial distribution and transaction sizes were exponentially distributed.

9.1. Relative Performance

Many strategies were tested in the simulations, but two emerged as preferable. Both were originally described in [Davi82]. They are:

(a) Davidson strategy #2. Iteratively break all 2-cycles by removing the node with lower weight.

(b) Davidson strategy #3. Break all 2-cycles by removing the node in partition 1. (As described in an earlier section, the performance of this heuristic can be improved by applying it to the smaller partition.)

If the precedence graph was not acyclic after all 2-cycles had been broken, a minimum-weight node removal strategy was used to finish the job.

Some of the other strategies tested include iteratively backing out the node of lowest weight in the smaller partition, backing out transactions in decreasing numerical order, and backing out the transaction with the highest degree-to-weight ratio in the precedence graph. All these strategies had performance essentially similar to Davidson #3, although they were slightly inferior to it. While Davidson #3 did not always give the best performance, it was typically as good or better than any of these other strategies on a given example.

The Davidson #2 strategy was distinctly different. It had better performance under low-backout conditions, but was worse under high-backout conditions. The point at which it became worse varied, but was usually around a backout rate of 15-20%. Since optimistic commit probably would be used under low-backout conditions, this strategy would be the one of choice. (There remains the possibility of using

[9] Except in experiments measuring the effects of changing this distribution.

Davidson's optimal 2-cycle breaking algorithm. However, since this has time complexity $O(n^3)$, it would be infeasible for large values of n. A faster approach would be to try several heuristics and use the best result.)

These results make it appear doubtful that there is much benefit to using a complex strategy. In an effort to answer these questions, a number of simulations were run with a "random" backout strategy, and with "antithetic" strategies. The "random" strategy, which was restricted to the smaller partition, simply made a random selection (uniform probability) among the nodes remaining. Antithetic strategies do the reverse of what a regular strategy does. For example, one such strategy removed nodes of greatest weight, rather than those of smallest weight.

The random strategy performed about as well as the others tested; its performance was slightly inferior to Davidson #3 on average, but in some cases it produced a better solution than any of the other strategies. The antithetic strategies were slightly worse than the random strategy and their "normal" counterparts, but the differences were slight. We conclude that in reduced graphs, the opportunities for heuristics to go awry are limited, and differences between any two heuristics are likely to be small.

One note about costs should be made. The backout rates in this report are simply the number of transactions backed out as a percentage of all transactions. This is not an accurate reflection of all environments. In particular, since backing out a readonly transaction does not require changes to the database, it may be more reasonable to give readonly transactions a cost of zero. Simulations run using this cost function showed relatively little change from the regular costs when backout rates were low. However, as the rates rose, the modified function had increasingly better performance, due to the number of readonly transactions that are in the dependency sets of backed-out update transactions.

9.2. Implementation Costs

An optimistic commit protocol is useless if it cannot be implemented efficiently. While the costs of the simulation need not reflect the costs in practice, there is no reason to think that they are not comparable. As an example of processor cost, simulations of sets of transactions with parameters $N1=N2=2000$, $M=50000$, $U=40\%$, $RO=80\%$, and $I=5$ had a backout rate of 0.4% (Davidson #2 strategy) and averaged 50 seconds of CPU time and about 500K bytes of main memory.[10] The reduced graphs contained an average of 35 nodes, an improvement of over 99%. (Only the modified Davidson #2 and #3 strategies were used.)

If we assume that the DDB could have run half the transactions submitted to it while the system was partitioned, and that the system can process 10 transactions per second, it would take 200 seconds of CPU time to process those transactions that could not be accepted during partitioning. In such a system, the optimistic approach would be highly effective.

[10]Programs written in C, run on a DEC VAX™ 11/780 under 4.1bsd UNIX™.