THE PERFORMANCE OF LOCKING PROTOCOLS IN DISTRIBUTED DATABASES

(Revised Version)

by

Ouri Wolfson

Technical Report #425

December 1986

* An extended abstract of this TR will appear at the 3rd International Conference on Data Engineering, February 1987, Los Angeles, CA
1. INTRODUCTION

1.1 Motivation

The field of distributed database performance evaluation is currently in a state of disarray. Performance results are inconclusive and sometimes contradictory, especially for concurrency control algorithms (a brief survey in [ACL] sustains this claim). We feel that an important reason for this is that many interrelated factors which affect performance (concurrency control, multiprogramming level, recovery algorithm, buffering scheme, data-distribution, etc.) have been studied as a whole multiuser system, without completely understanding the overhead imposed by each one. For example, [LN] indicates that the performance of a distributed concurrency control algorithm depends, among other things, on the load on the communication network, and the multiprogramming level. As we shall show; the concurrency control algorithm itself places a load on the network even in a single user system. We think that this load should be quantified first. Generally speaking, we suggest studying the overhead of each system component on each type of resource in a single user environment, and then using the results as a building block for further research.

The concurrency control algorithm interferes with the execution of transactions not only by placing a load on resources, but also by degrading resource utilization. This is also true even in a single user environment because, as we shall show, the concurrency control algorithm inhibits parallel execution of a transaction. This type of interference also has to be measured separately.

In this paper concurrency control by locking is used to demonstrate the approach that we propose. Specifically, each one of three locking protocols is first isolated and analyzed as a performance-overhead factor in the execution of a single transaction. Then, the combined overhead of the locking protocol and the two phase commit protocol is analyzed and determined, thus quantifying the total overhead incurred by controlling a single transaction.

1.2 Analysis of protocols

The main purpose of a locking protocol is to ensure correct interleaving of actions executed by different transactions. It does so by requiring that a set of rules be obeyed by the transactions. The rules generally impose some lock, unlock order. For example, two-phase-locking (2PL) states that a transaction has to obtain locks on all database entities it accesses, before unlocking any entity. In a centralized database the locking
protocol reduces concurrency, but the overhead incurred by a transaction as a result of obeying the rules is negligible. In a distributed database the overhead can be very significant for the following reason. Distributed transactions execute in parallel at different computers of the network (sites), and the processes at different sites communicate by message passing. Unfortunately, obeying the rules of the protocol may inhibit parallelism (thus inhibiting resource utilization) by requiring that certain steps be executed sequentially rather than in parallel. This requirement also necessitates message passing.

We assume that a transaction $T$ references a fixed set of entities dispersed among sites, and propose three measures for evaluating the overhead of a locking protocol. The first one deals with the issue of parallel execution. It measures the maximal number of protocol steps (i.e. locks, unlocks) that $T$ must execute sequentially in order to obey the protocol. This measure is most appropriate in a fast, and lightly loaded network, where the completion time of a transaction (or "response time") is determined by the longest sequence of steps, regardless of the number of intersite messages required in order to execute it. Since entities can be read or written only while locked, the longest sequence of lock-unlock steps is closely related to the longest sequence of disk accesses. The second measure also deals with parallel execution, and quantifies the longest sequence of intersite messages that $T$ must send in order to obey the protocol. It is most appropriate in a heavily loaded local-area network with the broadcast facility. One message sent from a site can be received by all other sites, but contention may delay placing of the message on the network; the number of times a transaction has to do so determines response-time. The third measure quantifies the total number of intersite messages that $T$ must send. It determines performance (or "throughput") in a heavily loaded wide-area network without the broadcast facility.

The introduced measures are used to analyze 2PL and two popular structured protocols. The first is a version of 2PL which also ensures deadlock freedom by imposing a total-order (structure) on the database entities. In addition to two-phase it provides the following rule. If the lock-request for an entity $x$ is issued by a transaction after the lock request of another entity, $y$, then $x$ succeeds $y$ in the total-order of entities ([UI, Section 12.6]). In other words, the order in which a transaction locks entities agrees with the structure. The second structured protocol we analyze is the tree protocol (TP) proposed in [SK]. It assumes that the database entities are physically or logically structured as a rooted directed tree, and allows more concurrency as a consequence. The reason is that some entities can be unlocked before all locks have been obtained. The protocol provides the following rules for locking and unlocking entities. A transaction issues its first lock on an arbitrary entity.
Subsequent locks can be issued when they request an entity which is a child in the database tree of some entity \( y \), and the transaction holds a lock on \( y \). Each entity can be locked only once. In addition to ensuring correctness, the tree protocol is also deadlock free. The overhead imposed by the structured protocols on a transaction depends on the particular way the referenced entities are structured. Therefore there is a range of possible overhead-measure values. We establish this range by determining its upper and lower tight bounds. The bounds are obtained by "worst" and "best" structures of referenced entities.

After formally defining the locking protocol overhead measures, we obtain the overhead of 2PL by each measure. Then, assuming a fixed distribution of the database entities, we obtain the overhead of TP for best and worst tree structures, and the overhead of deadlock-free 2PL for best and worst order on the database entities. We also demonstrate what these best and worst structures are, which is useful in deciding how to distribute the entities of a fixed database structure.

A locking protocol is usually used by a distributed transaction together with a commit protocol. The commit protocol ensures the 'all-or-nothing' effect of the transaction on the database. How do the results obtained about a locking protocol change when considering that it is combined with a commit protocol? We answer this question for the popular two-phase-commit (2PC) protocol. 2PC states that after a transaction agent executing at a site completes its computation, it executes a 'Ready' or 'Abort-Request' control step. If all agents are 'Ready', then each agent 'Commits', otherwise each agent 'Aborts'. Therefore, all the 'Ready' steps are executed before any 'Commit' step. This partial order causes a transaction to incur the same type of overhead it does in obeying a locking protocol. Moreover, a transaction can reduce the total overhead by properly ordering the control steps (Lock, Unlock, Ready/Abort-Request, Commit/Abort), and properly using control messages. For example, a control message from a site can indicate that all locks at the site have been obtained, and also that the Ready step has been executed. Indeed, in the final analysis it is discovered that the overhead of the locking and 2PC protocols combined, is almost identical to the locking-protocol overhead alone. This is true for all three measures.

1.3 Benefits of the analysis

In addition to demonstrating an approach, our results provide some immediate benefits, in the form of a decision tool that can be useful in choosing a locking protocol and in determining the database distribution, and
the control (steps) pattern of a transaction. Consider for example the control pattern. The analysis highlights a tradeoff between the longest sequence of messages and the total number of messages that exists when a transaction-control structure is determined. The tradeoff is precisely quantified, and the database-management-system (DBMS) transaction manager can use it by adjusting the structure according to the load on the communication network. This also highlights a (largely ignored) need for cooperation between the DBMS and the network manager (which monitors the communication load). The discussion section addresses these issues in further detail.

The analysis is also useful during the initial stages of building a communication network. At that point, an estimate of the average transaction overhead on the network is of crucial importance for capacity planning. Assuming that the access-pattern of an average transaction is known (e.g. entities at five sites), transaction-control overhead can provide a lower bound on the capacity needed for an average transaction.

1.4 Related work and paper outline

We are aware of very little work which is directly related to ours. For 2PL an informal discussion of overhead appears in [BG2]. However, even the basic problem of determining the minimum number of intersite messages required by a two-phase-locked transaction accessing $s$ sites, has not been addressed as far as we know. Some analysis of communication costs which is related to ours appears in [G, CP, DS], but it is restricted to commit protocols. For obtaining the results, a very general transaction model originally introduced in [KP1,KP2] is specialized, to represent locking and commit protocols, and the intersite messages they require. We regard this specializations as one of the contributions of this work.

The rest of the paper is organized as follows. Formal definitions necessary to consider the locking protocol are given in section 2, and the overhead measures are introduced in section 3. In section 4 we first concisely present all our results in tabular format, and then prove them. Section 5 extends the definitions and addresses the commit protocol effect on these results. Finally, section 6 is dedicated to a discussion of the results.
2. PRELIMINARY DEFINITIONS

2.1 Database and Transactions

A database is a finite set of physical entities. Entities are lockable units and we are not concerned with their granularity; they can be blocks, records, files, etc. In a distributed database a function called resides-at (or distribution) is associated with the set of entities. It assigns a value called site to each entity. A site is a computer-identification and obviously a physical entity cannot be stored in two computers.

Intuitively, a transaction represents the execution of program steps. The steps are executed by a set of transaction-"agents", each running at a site; the agents cooperate through message passing. A transaction leaves a consistent database, if it starts with a consistent one. We model a transaction by its locking protocol steps; all other steps (e.g., reads and writes) are ignored, since at this point we are interested in the overhead incurred as a result of obeying the locking protocol. The steps of a transaction are partially ordered, and the partial order is determined by two factors. First, the steps referencing entities residing at the same site are executed sequentially by the same computer. Second, a program of a transaction can wait for an intersite message from another program of the same transaction before executing the next step. This imposes an order between two steps of the same transaction executing at different sites.

Formally, a transaction $T$ is a directed acyclic graph (dag) $(S,A)$ such that the set of arcs $A$ is partitioned into two sets: a set of intrasite arcs $A_1$, and a set of intersite arcs $A_2$. $S$ is a set of Lock, Unlock steps referencing the database entities; locks are exclusive. The lock of an entity $x$, denoted $L_x$, is in $S$ if and only if its unlock, $U_x$, is in $S$. We say that $L_x$ and $U_x$ are executed at the residence site of $x$. $A_1$ is a set of arcs $(a,b)$ such that $a$ and $b$ are executed at the same site. $A_2$ is a set of arcs $(a,b)$ such that $a$ and $b$ are executed at different sites. $A_1$ induces in $T$ a partial order in which all steps executed at the same site are totally ordered. In this partial order $L_x$ precedes $U_x$ for any entity $x$ referenced in $T$. These restrictions on $A_1$ give in the case of one site the usual model of centralized transactions as sequences of steps. $A_1$ represents the order in which the steps are executed at each site. $A_2$ represents the intersite messages sent by the transaction, in the following sense. Assume that $a$ is executed at site 1 and $b$ is executed at site 2. $(a,b)$ is an arc in $A_2$ if and only if: between $a$ and its immediate successor at site 1, a message was sent from site 1 to site 2, and the message was received at site 2 before $b$, but after its immediate predecessor at site 2.
The database and transaction definitions are similar to the ones introduced in [KP1] with the exception of the distinction between intersite and intrasite messages. This distinction is crucial for the upcoming analysis. A trivial observation which we use extensively in proving our results is the following. If in a transaction, a step executing at one site precedes a step executing at another site then every path between the two steps contains at least one intersite arc.

2.2 Locking Protocols

A locking protocol is a set of transactions. Intuitively, the set consists of the transactions resulting from a possible execution of programs obeying some rules on the locking and unlocking of entities. We now formally define the locking protocols considered in this paper. The first is two-phase-locking: $2PL = \{ T | \delta T \text{ is a transaction in which every lock precedes all the unlocks} \}$. (A two phase locked transaction has at least the precedences of the transaction in Fig. 1(a).)

The second locking protocol is deadlock free two phase locking. It achieves deadlock freedom by imposing an order, $p$, on the database entities, and will be denoted $DF2PL_p$. $DF2PL_p = \{ T | T \text{ is in 2PL and: } Lx \text{ precedes } Ly \text{ in } T \text{ implies } x \text{ precedes } y \text{ in } p \}$. (Fig. 1(b).)

For a database which is structured as a rooted directed tree $t$, the tree policy, denoted $TP_t$, is defined as follows: $TP_t = \{ T | T \text{ is a transaction in which there exists a lock step which precedes all other steps of } T, \text{ and any other lock step, } Ly, \text{ is preceded by the lock of the father of } y \text{ in } t, f(y), \text{ and succeeded by the unlock of } f(y) \}$. (Fig. 2.)

For the sake of brevity, we omit the formal definitions of schedule, serializability and deadlock freedom (see [KP1], [W], [WY]). However we will observe that $2PL$ ensures serializability ([EGLT]), while $DF2PL_p$ and $TP_t$ ensure serializability and deadlock freedom (see [H,U] and [SK] respectively). The set $2PL$ we defined is maximal in the following sense. If some transaction which is not in $2PL$ is added to the set, then there exists some non-serializable schedule (or history) of transactions in the newly created set. Similarly, if for some total order $p$ a transaction is added to $DF2PL_p$ then there exists some non-serializable schedule of transactions in the newly created set, or some partial schedule resulting in a deadlock. The same is true for the tree protocol. These "nice" properties continue to hold for $2PL$ and $DF2PL$ if the set of transactions is defined as above, and shared locks are also allowed by the definition of a transaction (not true for TP). Therefore, the results that we
In this section we formally define the overhead measures, and discuss them. First we define the measures for a transaction, then we generalize the definitions to the locking protocols, and show how to take the database structure into account.

For a transaction $T$, the execution length, denoted $El(T)$, is the number of vertices on the longest path in $T$. The message path of $T$, denoted $Mp(T)$, is the maximum number of intersite arcs on a path in $T$. The total number of

Figure 1 - Example of a transaction in 2PL (a) and in $DF2PL_p$ (b). The entity referenced by a step is omitted. The execution site of the step is indicated by its subscript.

Figure 2 - A database structure $t$ (a) and a transaction in $TP_i$ (b). Entities $v, x, z$ reside at different sites.

obtain about 2PL and DF2PL hold without the "exclusive-locks only" restriction.

3. THREE OVERHEAD MEASURES

In this section we formally define the overhead measures, and discuss them. First we define the measures for a transaction, then we generalize the definitions to the locking protocols, and show how to take the database structure into account.
of messages in $T$, denoted $Nm(T)$, is the number of intersite arcs in $T$.

We isolate the locking protocol overhead by fixing a set of entities, and considering the minimum overhead measure value among all transactions of the protocol referencing the set. Formally, for a protocol $P$ and a set of entities $X$ we denote by $P(X)$ the set of transactions which belong to $P$ and reference exactly the set of entities $X$. We define the execution-length-overhead of protocol $P$ on set of entities $X$, $EIP(X)$, to be $\min_{T \in P(X)} \{EI(T)\}$. The intuitive meaning is that any transaction which obeys the rules of the protocol, and accesses exactly $X$, will execute at least $EIP(X)$ Lock-Unlock steps sequentially. Similarly, we define $Mp.P(X) = \min_{T \in P(X)} \{Mp(T)\}$ and $Nm.P(X) = \min_{T \in P(X)} \{Nm(T)\}$. They are the message-path-overhead and message-number-overhead respectively.

The previous definitions suffice to measure the overhead of 2PL. However, the set of transactions in a structured protocol depends on the particular database structure. For example, if $t_1 \neq t_2$ then $TP_{t_1} \neq TP_{t_2}$. As a consequence, different structures can have different values for each measure. In other words, a structured protocol has a set of values for each measure; we will determine are the minimum and the maximum value in each set. What do they mean? For example, $\max_{T \in P(X)} Nm.TP_{t}(X)$ is the maximal number of intersite messages required by the protocol regardless of the particular tree structure; i.e., there is no database tree for which every transaction referencing $X$ and obeying the protocol needs more messages. This value is obtained when $X$ is structured as a certain tree. We call it a worst tree by the $Nm$ measure. Similarly, we define a worst tree by the other two measures, and a worst sequence of database entities for DF2PL by each measure.

The minimal value in a set is interesting for similar reasons. For example, $\min_{T \in P(X)} Nm.TP_{t}(X)$ is the minimal number of messages required by the tree protocol regardless of the tree structure. A tree $t$ for which this minimum is obtained is a best, or optimal, tree by the $Nm$ measure. Similarly a best tree is defined by the other two measures, and a best sequence is defined for DF2PL by each of the three measures.

In the course of proving our results in the next section, we demonstrate a structure for DF2PL which is best by all three measures, and one which is worst by all three measures. We do the same for TP, with one exception. The worst structure by the $EI$ measure is different from the worst structure by the other measures (see Observation 5 in the next section).
4. RESULTS

In this section, we first concisely state our results (see RESULTS-TABLE below) and then prove each one. The entries in RESULTS-TABLE are numbered 1-15. Each entry corresponds to an overhead-measure and a locking protocol, and provides a value with respect to a fixed set of entities, X. The value is provided as a function of the distribution parameters of X. The distribution parameters of X are $k_1,...,k_s$ for $s > 1$, meaning that $k_1$ entities of X reside at site one, $k_2$ reside at site two, etc.

The rest of this section is dedicated to proving the results-table entries. The proof of an entry $i$ corresponding to protocol $P$ and measure $M$ consists of two parts: existence and bound. The existence proof for 2PL demonstrates a transaction of $2PL(X)$ for which the value of $M$ is the one given in the entry. For the structured protocols it demonstrates a database structure $st$ and a transaction in the protocol structured by $st$. The bound proof for 2PL shows that the value of $M$ for any transaction in $2PL(X)$ is at least the one given in the entry. If $P$ is a structured protocol and the entry corresponds to min $M.P$, then the proof shows that the value of $M$ for any transaction in $P(X)$ is at least the one given in the entry, regardless of the structure on X. If the entry corresponds to max $M.P$, then the proof shows two things. First, that there does not exist a transaction $T$

<table>
<thead>
<tr>
<th>Protocols</th>
<th>2PL two-phase-locking</th>
<th>$DF^2PL_p$ deadlock-free 2PL</th>
<th>$TP_t$ tree protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overhead Measures</td>
<td>min $p$</td>
<td>max $p$</td>
<td>min $i$</td>
</tr>
<tr>
<td>$El$ longest sequence of protocol steps</td>
<td>$2\max_k i$</td>
<td>$\Sigma k_i + \max k_i$</td>
<td>$\Sigma k_i + \max k_i$</td>
</tr>
<tr>
<td>$Mp$ longest sequence of intersite messages</td>
<td>1</td>
<td>s</td>
<td>$A+1$</td>
</tr>
<tr>
<td>$Nm$ total number of intersite messages</td>
<td>2$s$-2</td>
<td>2$s$-2</td>
<td>$A+\delta 1$</td>
</tr>
</tbody>
</table>

RESULTS-TABLE: Overhead-measures of locking protocols on a set of entities X. $k_i$ entities of X reside at site $i$, for $1 \leq i \leq s$ and $s > 1$. $A$ represents the expression $\min \{2(\Sigma k_i - \max k_i), \Sigma k_i - 1\}$. 
in the protocol structured by \( st \) of the existence proof, such that \( M(T) \) is lower than the value in the entry.

Second, that for any structure \( d \) there exists a transaction \( T \) in the protocol structured by \( d \), such that \( M(T) \) is the value given in the entry or less. In other words, we show that there exists a database structure for which the protocol requires that the value of the measure be at least the one in the entry; and the value in the entry is sufficient in order to obey the protocol, regardless of the structure. For example, for entry 15, this proves the following. There exists a tree structure which necessitates \( A+s-1 \) intersite messages for referencing the set of entities \( X \), and obeying the tree protocol. Also, \( A+s-1 \) intersite messages suffice for referencing the set \( X \) and obeying the protocol, regardless of the particular tree structure.

**Proofs of Entries in the RESULTS-TABLE**

For the first proof the following notation is needed. Let \( T^0 \) be some transaction in \( 2PL(X) \) which has an intersite arc from the last lock at every site to the first unlock at each other site, and no other intersite arcs (Fig. 1(a)). Note that there is a whole subset of \( 2PL(X) \) with the specified intersite arcs, one for each order of locks and unlocks at each site (that is the reason we defined \( T^0 \) as "some" transaction in \( 2PL(X) \)).

**Proof of entry number (1)** \( EI.2PL = 2\max_k \) : (Existence) The transaction is \( T^0 \).

**(Bound):** By definition, any transaction referencing \( X \) has a path of length \( 2\max_k \), or more. \( \square \)

**Proof of entry number (2)** \( Mp.2PL(X) = 1 \): (Existence) The transaction is \( T^0 \).

**(Bound):** Any transaction of \( 2PL(X) \) has at least one intersite arc. \([\] \)

Let \( p^+ \) be some total order of the database entities in which every entity at site \( i \) precede all entities at site \( i+1 \) for \( i = 1, ..., s-1 \). We will show later that \( p^+ \) is a best structure for DF2PL; at this point we need it for the following definition. Let \( T^+ \) be some transaction in \( DF2PL_{p^+}(X) \) which has exactly the following intersite arcs:

1. from the last lock at site \( i \) to the first lock at site \( i+1 \) for \( i = 1, ..., s-1 \),
2. from the last lock at site \( s \) to the first unlock at site \( i \) for \( i = 1, ..., s-1 \).

\( T^+ \) has the format of Fig. 1(b). Clearly, there exists at least one such transaction in \( DF2PL_{p^+}(X) \). There can be many transactions of \( DF2PL_{p^+}(X) \) with the specified intersite arcs, each unlocking entities in a different partial order.
Proof of entry number (3) $Nm.2PL(X) = 2s-2$: (Existence) The transaction is $T^+$.

(Bound): Assume that $T \in 2PL(X)$. Let $T$ have $l$ unlock steps of which none succeeds another unlock step; and $m$ lock steps of which none precedes another lock step. Clearly $s \geq l$, $m \geq 1$. The total number of arcs between these steps is $l \cdot m$, because all locks precede all unlocks. At most $\min(l,m)$ of these arcs are from a lock to an unlock at the same site. Thus there are at least $l \cdot m - \min(l,m)$ intersite arcs from locks to unlocks.

Let $r_1, ..., r_s$ be the first unlock steps executed at sites $1, ..., s$ respectively. Since only $l$ unlocks do not have an unlock predecessor, $s-l$ of these steps are preceded by an unlock step executing at a different site. Thus, the number of intersite arcs between two unlocks in $T$ is at least $s-l$. Similarly, it can be shown that the number of intersite arcs between two locks is at least $s-m$. Thus the total number of intersite arcs is at least $2s-\min(l,m)$. It is easy to see that $l+m+\min(l,m)-l \leq 2s-2$ for any positive integers $l$ and $m$, therefore $T$ has at least $2s-2$ intersite arcs. []

Note that both existence proofs, for $EI.2PL(X)$ and for $Mp.2PL(X)$, use $T^0$. A different-format transaction, $T^+$, is used in the existence proof for $Nm.2PL(X)$.

Proof of entry number (4) $\min_{p} EI.DF.2PL_p = \sum_{i} k_i + \max_{i} k_i$: (Existence) The structure is $p^+$ and the transaction is $T^+$.

(Bound): Regardless of the total order of database entities, $p$, if $T \in DF.2PL_p(X)$ then $T$ has $\sum_{i} k_i$ totally ordered locks which precede $\max_{i} k_i$ totally ordered unlocks. Thus $EI(T) \geq \sum_{i} k_i + \max_{i} k_i$. []

Proof of entry number (5) $\min_{p} Mp.DF.2PL_p(X) = s$: (Existence) The structure is $p^+$ and the transaction is $T^+$.

(Bound): Let $p$ be some total order of the database entities and $T \in DF.2PL_p(X)$. Assume, without loss of generality, that the first lock that $T$ executes is at site 1. Denote by $d_i$ the first lock executed by $T$ at site $i$ for $i=2, ..., s$. Since all locks are totally ordered each $d_i$ has an intersite arc incoming from the lock which immedi-
ately precedes it in $T$. Denote this intersite arc entering $d_i$ by $l_i$. Also, denote by $m$ a site different than the site at which $T$ executes its last lock. There is an intersite arc, $l_{i+1}$ entering the first unlock at site $m$. $l_2, ..., l_x, l_{x+1}$ are on a path. Thus $Mp(T) \geq s$. []

**Proof of entry number (6)** $\min_p Nm. DF2PL_p(X) = 2s - 2$: (Existence) The structure is $p^+$ and the transaction is $T^+$.

(Bound): If $p$ is a total order of database entities and $T \in DF2PL_p(X)$ then $T \in 2PL(X)$ thus the proof follows from entry number 3. []

Before proceeding we need the following preliminaries. For the set of database entities $X$, *a sequence on $X$* is a total ordering of the entities of $X$. A *site alternation* of a sequence on $X$ is a pair of adjacent entities which reside at different sites. We are interested in a sequence on $X$ which has the maximum number of site alternations, and will prove that algorithm MAXALTER (Fig. 3) builds such a sequence. Given $X$, we denote the output-sequence of the algorithm by $p^-(X)$. We will show that it is a worst sequence for $DF2PL(X)$ by all measures. It is also a worst tree for $TP(X)$ by the $Mp$ and $Nm$ measures.

The algorithm MAXALTER assumes without loss of generality that the distribution parameters of $X$ are $k_1 \leq \ldots \leq k_s$. It builds $p^-(X)$ sequentially by appending one entity at a time at the end of $p^-(X)$, and removing it from $X$. It is done at steps 3, 5 or 8 of MAXALTER. Note that MAXALTER is nondeterministic in the sense that at these steps we just append an entity from some site, without specifying which entity. However, this is immaterial, because for our purposes only the residence site of the entity matters.

Intuitively, MAXALTER works as follows. While possible, a site alternation is created each time an entity is appended. The attempt is to leave $X$ with an equal number of entities at sites 1 and 2 (loop at steps 1-4). If successful, the second loop (steps 6-10) is performed. Otherwise, it means that $k_1 > \sum_{i=2}^{s} k_i$ and the algorithm ends after performing step 5. Only at step 5 an entity can be appended without creating a new site alternation of $p^-(X)$.

*Example of MAXALTER:* Assume that entities $x_1, x_2, \ldots, x_3$ reside at site 1, entities $y_1, y_2, y_3$ reside at site 2, and entities $z_1, z_2, z_3$ reside at site 3. Then the sequence built by MAXALTER is $x_1, x_2, z_2, z_3, y_1, y_2, y_3, x_3$. If at site 1 there are also entities $x_6, \ldots, x_{10}$ then the sequence built is $x_1, z_1, x_2, z_2, z_3, x_3, y_1, x_5, y_2, x_6, y_3, x_7, \ldots, x_{10}$. ☐
MAXALTER(X);
1. Do while X has some entities at site 2;
2. If X has the same number of entities at sites 1 and 2 then go to 6;
3. Append at the end of \( p^-(X) \) (and remove from X) an entity from site 1, then an entity from the highest site at which X has entities;
4. end;
5. Append at the end of \( p^-(X) \) all entities left at site 1 in an arbitrary order; quit;
6. Do while X is not empty;
7. For \( i=1 \) to \( s \) do;
8. If X has some entities at site \( i \) then append at the end of \( p^-(X) \) (and remove from X) an entity from site \( i \);
9. end;
10. end;

Figure 3 - Algorithm MAXALTER totally orders a set of input entities X into a sequence \( p^-(X) \). \( k_i \) entities of X reside at site \( i \) for \( 1 \leq i \leq s \), and we assume \( k_1 \geq \ldots \geq k_s \).

Lemma 1: For a set of entities X with distribution parameters \( k_1 \geq \ldots \geq k_s \) the number of site-alternations of \( p^-(X) \) is \( \min\{2(\sum_{i=2}^{s}k_i - \max k_i), \sum k_i - 1\} \).

Proof: In algorithm MAXALTER each execution of step 3 creates two site alternations (except the first one, of course, which creates only one). Assume that \( k_1 > \sum_{i=2}^{s}k_i \), or in other words \( 2(\sum_{i=2}^{s}k_i - \max k_i) \leq \sum k_i - 1 \). Then step 3 will be executed \( \sum_{i=2}^{s}k_i \) times and the "go to" at step 2 will never be performed. The algorithm will end at step 5, which will create one more site alternation. Thus the total number of site alternations of \( p^-(X) \) will be \( 2(\sum_{i=2}^{s}k_i - \max k_i) \).

Assume now that \( k_1 \leq \sum_{i=2}^{s}k_i \). Then entities are appended to \( p^-(X) \) only at steps 3 and 8 and at these steps the appendix of each entity creates one site alternation. Thus the total number of site-alternations in \( p^-(X) \) is equal
Lemma 2: For a set of entities $X$ with distribution parameters $k_1 \geq \ldots \geq k_n$, $p^-(X)$ has the maximum number of site-alterations among all sequences on $X$.

Proof: Obviously a sequence $g$ on $X$ has at most $\sum_{i=1}^{n} k_i - 1$ site alternations. Assume now that $k_1 > \sum_{i=2}^{n} k_i$. Then at most $\sum_{i=2}^{n} k_i$ entities residing at site 1 can be immediately preceded in $g$ by an entity which does not reside at site 1. Thus the total number of entities which can be immediately preceded by an entity residing at a different site is at most $2\sum_{i=2}^{n} k_i$. []

Given a total order $p$ of the database entities, denote by $p(X)$ the subsequence of $p$ consisting of the entities of $X$. Also, denote by $T(p)$ some transaction in $DF2PL_p(X)$ which has exactly the following intersite arcs:

1) From $L_x$ to $L_y$ if $x, y$ is a site alternation in $p(X)$.
2) From the last lock executed by $T(p)$ to each first unlock it executes at a different site (than the last lock).

Again, $T(p)$ has the format of Fig. 1(b). Clearly, there exists at least one such transaction in $DF2PL_p(X)$.

Observation 1: $M_p(T(p))$ is equal to the number of site alternations of $p(X)$ plus one.

Observation 2: For any transaction in $DF2PL_p(X)$ the $M_p$ measure is at least $M_p(T(p))$.

Denote by $p^-$ some total order of the database entities which has $p^-(X)$ as a subsequence.

Proof of entry number (7) $\max_{p} El_{DF2PL_p} = \sum_{i} k_i + \max_i$: (Existence) The structure is $p^-$ and the transaction is $T(p^-)$.

(Bound): By entry number 4 there is no transaction in $DF2PL_p(X)$ with a lower $El$ value. For any total order of database entities $p$ and any $T \in DF2PL_p(X)$ in which there are no intersite arcs between two unlocks $El(T) = \sum_{i} k_i + \max_i$. []

Proof of entry number (8) $\max_{p} M_p DF2PL_p = A + 1$: (Existence) The structure is $p^-$ and the transaction is $T(p^-)$.

The proof follows from Lemma 1 and Observation 1.
The proof follows from Lemma 2 and Observations 1 and 2. []

Observation 3: \( Nm(T(p)) \) is equal to the number of site alternations of \( p(X) \) plus \( s-1 \).

Observation 4: For any transaction in \( DF2PL_p(X) \) the \( Nm \) measure is at least \( Nm(T(p)) \).

Proof of entry number (9) \( \max Nm_{DF2PL_p} = A+s-1 \): (Existence) The structure is \( p^* \) and the transaction is \( T(p^*) \). The proof follows from Lemma 1 and Observation 3.

(Bound): The proof follows from Lemma 2 and Observations 3 and 4. []

For the next proofs we assume again that \( k_1 \geq \ldots \geq k_s \) and we structure the set of database entities \( X \) as a directed tree \( t \) as follows. Arbitrarily structure the entities at site \( i \) as a directed tree rooted at \( r_i \) for \( i = 1, \ldots, s \). Build \( t \) by making \( r_2, \ldots, r_s \) the sons of \( r_1 \). \( t \) is illustrated in Fig. 4. Denote by \( t^* \) a directed tree structure of the database entities, of which \( t \) is a subtree. We will show that \( t^* \) is a best structure for \( TP_i(X) \) by all measures. Let \( F \) be some transaction of \( TP_i(X) \) which has exactly the following intersite arcs: from \( Lr_i \) to \( Lr_j \) and from \( Lr_j \) to \( Ur_i \) for \( i = 2, \ldots, s \). \( F \) is illustrated in Fig. 5.

Proof of entry number (10) \( \min i.El.TP_i = 2\max_i +1 \): (Existence): The tree is \( t^* \) and the transaction is \( F \). (We assume that the number of entities at sites 1 and 2 are equal; otherwise the value for the entry is \( 2\max_i \)).

Figure 4: Optimal structuring of a set of entities into a directed tree.
Figure 5: Optimal transaction obeying the tree protocol, by the El and Mp measures.

(Bound): Trivial if $X$ has $\max k_i$ entities at one site only. Otherwise, let $T \in TP_i(X)$ for some tree $i$. There exists a site at which $T$ executes $2\max k_i$ steps at which the root of the tree referenced by $T$ does not reside. The first lock executed by $T$ at that site is preceded by another step, thus $El(T) \geq 2\max k_i + 1$. 

Proof of entry number (11) $\min_i Mp.TP_i = 2$: (Existence) The tree is $i^*$ and the transaction is $F$.

(Bound): Let $T \in TP_i(X)$ for some tree $i$. Consider $Ly$, the first lock executed by $T$ at site at which the root of the subtree referenced by $T$ does not reside. Since the father of $y$ resides at a different site there exists an intersite arc entering $Ly$ and an intersite arc exiting $Ly$ or a step succeeding $Ly$. These two arcs are on a path. 

Let $G$ be some transaction of $TP_i(X)$ which has exactly the following intersite arcs: from $Lr_i$ to $Lr_{i+1}$ for $i=1,\ldots,s-1$ and from $Lr_s$ to $Ur_1$ (Fig. 6).

Proof of entry number (12) $\min_i Nm.TP_i = s$: (Existence) The tree is $i^*$ and the transaction is $G$.

(Bound). Consider some transaction $T \in TP_i(X)$ for some tree $i$. The unlock of the root of the subtree on $X$ (or a predecessor executing at the same site) must have an entering intersite arc. The first lock at any other site must also have an entering intersite arc. Thus $T$ has at least $s$ intersite arcs. 

Proof of entry number (13) $\max_i El.TP_i = \sum_i k_i + \max k_i$: (Existence) Assume without loss of generality that $\max k_i$ entities of $X$ reside at site $s$. The structure is some total order in which every entity at site $i$ precedes all
entities at site \(i+1\), for \(i = 1, \ldots, s-1\). Denote it by \(T^-\). We show that it is worst by the \(El\) measure. The transaction from \(TP_r(X)\) is one with the format of Fig. 1(b) (formally, \(T(T^-)\)).

(Bound): If \(T \in TP_r(X)\) then the first lock executed by \(T\) at site \(s\) precedes \(2 \max_k - 1\) totally ordered steps, and is preceded by \(\sum_{i} k_i - \max_k\) totally ordered lock steps, thus \(El(T) \geq \sum_{i} k_i + \max_k\). For an arbitrary tree \(I\), there exists a transaction with the format of Fig. 1(b) in \(TP_r(X)\); its \(El\) measure is \(\sum_{i} k_i + \max_k\). [1]

**Proof of entry number (14) \(\max_{T \in TP_r(X)} M_p TP_r = A + 1\): (Existence)** Let the tree be the total order of database entities \(p^-\) defined before. Let the transaction in \(TP_r(X)\) be \(T(p^-)\).

(Bound): Let \(T\) be some transaction in \(TP_r(X)\). All locks of \(T\) are totally ordered and any path between \(L_x\) and \(L_y\) has an intersite arc if \(x, y\) is an alternation of \(p^-\). By Lemma 1, \(T\) has at least \(A+1\) intersite arcs on a path. Now let \(I\) be an arbitrary tree structure of the database entities. Define \(p\) to be an arbitrary extension of \(I\) (total order corresponding to \(I\)). Clearly \(T(p)\) is in \(TP_r(X)\), and by Lemma 2 it has no more than \(A+1\) intersite arcs on a path. [1]

**Observation 5** Note that \(T^-\) is not worst by the \(M_p\) measure. For example, if \(T^-\) is the sequence \(x, y, z\), with \(x\) residing at site 1 and \(y, z\) residing at site 2, then we can define a transaction referencing \(x, y, z\) in which the maximum number of messages on a path is two. On the other hand, if \(z\) resides at site 1 instead of site 2, we cannot define such a transaction. Similarly it can be shown that \(p^-\) is not worst by the \(El\) measure.

**Proof of entry number (15) \(\max_{T \in TP_r(X)} N_m TP_r = A + s - 1\): (Existence)** The tree is the total order of database entities \(p^-\):

\[
\begin{array}{cccc}
\text{site-1} & \text{site-2} & \text{site-3} & \text{site-s} \\
Lr_1 & \rightarrow & Lr_2 & \rightarrow & Lr_3 \rightarrow \cdots \rightarrow Lr_s \\
U_r_1 & \rightarrow & \cdots & \rightarrow & \cdots \\
\end{array}
\]

Figure 6: Optimal transaction obeying the tree protocol, by the \(Nm\) measure.
and the transaction is $T(p^*)$.

(Bound): Let $T$ be some transaction in $TP_p(X)$. By Lemma 1, $T$ has at least $A$ intersite arcs which enter a lock or a preceding unlock executed at the same site. Denote by $Lx_i$ the last lock step executed by $T$ at site $i$, for $i = 1, \ldots, s$. $s-1$ of the $x_i$'s immediately precede in $p^*(X)$ an entity residing at a different site. For $x_i$ denote this entity by $y_i$. $Ly_i$ precedes $Ux_i$ and succeeds $Lx_i$ in $T$. Therefore in $T$ there exist at least $s-1$ intersite arcs with the following property: each one enters an unlock which does not precede any lock step executed at the same site. Overall, $T$ has at least $A+s-1$ intersite arcs. Now let $t$ be an arbitrary tree structure of the database entities. A transaction $T \in TP_t(X)$ for which $Nm(T) \leq A+s-1$ is identified in the same way as in the bound proof of entry number 14.

5. INCORPORATING THE TWO-PHASE-COMMIT PROTOCOL

In this section we extend the transaction model to include the two phase commit protocol steps. A committed transaction in the extended model is represented by a partially ordered set of control steps. In addition to the lock-unlock control steps, the transaction also executes a $R_i$ and a $C_i$ control step at each site $i$ at which it references entities. They represent the 'site i Ready' and 'site i Commit' steps respectively. $R_i$ succeeds the last lock step at site $i$, and every $R_j$ step precedes all $C_i$ steps. We still require that all the control steps executed at a site are totally ordered. In [G], Gray discusses the centralized and linear communication structure of the 2PC protocol and in [CP], the hierarchical and distributed structures are also discussed. The extended model can represent all of them. For example, in a centralized communication structure using site $j$ as a coordinator, $R_j$ precedes all other $R_i$'s, and all $R_i$'s precede $C_j$, which in turn precedes all other $C_i$'s. In a linear communication structure the commit protocol control steps are totally ordered $R_1 \rightarrow R_2 \rightarrow \cdots \rightarrow R_n \rightarrow C_n \rightarrow C_{n-1} \rightarrow \cdots \rightarrow C_1$.

In a distributed communication structure all $R_i$'s are pairwise incomparable, and so are all $C_i$'s.

With this extended transaction model the only changes in the results table are: 1) entry number 12 becomes $2s-2$, and 2) the values for the $El$ measure increase by the constant 2 (accounts for the $R_i$ and $C_i$ steps that must be totally ordered at a site). Concerning entry number 12, note that the number of intersite messages required by a transaction executing control steps at $s$ sites is now at least $2s-2$. The proof, based on the restriction that every $R_j$ precedes all $C_i$'s is similar to the one given for entry number 3 in Section 5. The bound proofs for all other entries are almost identical to the ones given in Section 5, thus omitted. The existence proofs
and optimal partial ordering of control steps by different measures are obvious when examining figures 7-10.

**Figure 7** - Optimal 2PL transaction by the EI and Mp measures. Subscripts of the steps represent their execution sites.

**Figure 8** - Optimal DF2PL transaction by the Mp and Nm measures. It is also optimal by the Nm measure for 2PL and TP.
In this paper we proposed a "divide and conquer" approach to performance evaluation of distributed databases; it isolates and analyzes single components in relatively simple environments. We demonstrated this approach for the locking and commit mechanisms. Specifically, we introduced three measures of the overhead imposed by a protocol on transactions in a distributed database. The first one quantifies the longest sequence of necessary protocol steps ($El$); the second, the longest sequence of necessary intersite messages ($Mp$); and the third, the total number of necessary intersite messages ($Nm$). Then we analyzed each one of three important locking protocols, by each measure: two-phase-locking (2PL), and the structured protocols deadlock-free-two-
phase-locking (DF2PL) and the tree-protocol (TP). The analysis provides for each pair (locking-protocol, measure) a lower bound on the overhead, for any transaction obeying the protocol. For the structured protocols there exists a range of such lower bounds, depending on the particular structure imposed on the database entities. The upper and lower bounds of this range are presented. All bounds are tight. Then we showed that the bounds remain almost unchanged when a locking protocol is combined with the two phase commit protocol. (The reason for this is that precedence information for both protocols can be carried by the same control message.) The optimal interleaving of locking and commit protocol steps, by the three measures, was also identified.

6.1 Tradeoffs

The measures we provide quantify some intuitive tradeoffs. For example, it is intuitive that the more "spread-out" the set of entities referenced by a transaction, the better its parallelism (i.e., lower overhead by the El measure); however, this tends to increase the number of required intersite messages (i.e., higher overhead by the Nm measure). The results point out some tradeoffs which are not so obvious. For example, assume that the database management system enforces 2PL. The total number of intersite messages can be decreased at the expense of increasing the number of sequential intersite messages, and a lower transaction parallelism (and vice versa). Specifically, if locking is done by a transaction in parallel at the different sites, then the total number of intersite messages required increases. The same is true if the (commit protocol) Ready steps are executed in parallel. On the other hand, if the transaction obtains all locks and executes the Ready step at one site before starting to issue lock requests at another site, then the total number of intersite messages required decreases.

In a local-area network with the broadcast facility, the total number of point-to-point messages is not an important factor, assuming that a message which a site sends on the network, can be received by every other site. Thus parallel execution of control steps is preferable. In a wide area network, depending on the load, the total number of intersite messages may or may not be a bottleneck. The transaction manager can be designed to dynamically adjust to changing conditions by decreasing overhead according to one measure, while increasing it according to another. Similar tradeoffs exist for the tree protocol.
6.2 Comparison of Protocols

2PL is at least as good as the structured protocols. The inferiority of the structured protocols compared to 2PL is related to the fact that they are deadlock free and therefore they require a more "dense" partial ordering of protocol steps, which in turn increases overhead. However, it should be considered that when 2PL is used there is an additional overhead of deadlock detection and recovery. In order to establish whether 2PL or a deadlock free protocol is better, the overhead of the various deadlock detection and deadlock resolution schemes should also be measured and analyzed, as should the combined overhead of transaction control with such schemes. (When analyzing the combined overhead we would not be surprised to discover that by properly scheduling deadlock detection, the mechanism can use transaction control messages.)

We can also observe that TP is at least as good as DF2PL by all measures (i.e. the upper bound on the overhead is identical but the lower bound is smaller). The superiority can also be attributed to a disadvantage that TP has when compared to DF2PL. It is that tree protocol transactions cannot access every subset of database entities, therefore they are allowed a more "sparse" partial ordering of locks and unlocks.

It is also interesting to note that given a good distribution of the database structure (see next subsection), DF2PL and 2PL impose the same overhead by the $N_m$ measure; thus, deadlock freedom is obtained at no additional intersite messages. In other words, assume that we have a choice between using 2PL and DF2PL as the concurrency control algorithm; additionally, assume that the purpose is to minimize intersite messages (possibly at the expense of transaction parallelism). Then, DF2PL with the total order proposed in the next subsection should be used. It will not require more intersite messages than 2PL, but will save the overhead of deadlock detection and resolution.

6.3 Distributing a Database Structure

Assuming a fixed database structure and no a priori information concerning the frequency at which sets of entities are accessed by transactions, the analysis indicates the following. The least overhead is imposed by the tree protocol if each site is allocated a whole subtree, with the entities at different sites incomparable in the partial order defined by the directed tree. For DF2PL an optimal distribution of the database sequence is one in which all entities at one site precede all entities at another site.
6.4 Future Work

This research should be carried further in several directions. First, it is interesting to see whether the tools we introduced here can be used to determine the extent to which locking inhibits resource utilization when several transactions are run, and conflicting locks, in addition to order requirements, increase the "longest sequence". Additionally, other components, some which interact with the locking mechanism (e.g. deadlock detection, query processing), and some which may run instead of the locking mechanism (i.e. other concurrency control algorithms), should be subjected to the same type of analysis. Also, isolated loads on different types of resources (I/O, CPU, etc.) should be measured, before studying the behavior of components as a whole multiuser system.

ACKNOWLEDGEMENT: I wish to thank Ephraim Korach for helping establish entries 3 and 8 in the results-table, Shimon Even for proposing the execution-length measure, and the referees for helpful comments.

REFERENCES


