CUCS-205-85

An Automated Performance Analysis of A Two Phase Locking Protocol*

Nihal Nounou and Yechiam Yemini

Computer Science Department
NY, NY, 10027

August 1985

ABSTRACT

ANALYST, an interactive protocol performance analyzer, is used to analyze the performance of a two phase locking protocol. ANALYST implements a specification-based methodology for performance analysis of protocols which extracts from an algebraic specification of a protocol a model of its timing behavior. Any timing requirement or performance measure that can be formally specified in terms of attributes of this timing behavior can be thus analyzed. An algebraic specification of a two phase locking protocol that uses time-out for deadlock detection is provided. Two timing requirements necessary for its efficient performance are specified and analyzed yielding optimal settings of protocol parameters (such as time-out rate). Additionally, the mean response time and probability of deadlock of the protocol are specified and analyzed. This, to the best knowledge of the authors, is the first automated, analytic performance analysis of such a high-level protocol.

*This research was supported in part by the Defense Advanced Research Projects Agency under contract N00039-84-C-0165, the New York State Center for Advanced Technology in Computers and Information Systems under NYSSTF CAT(83)-8, and a grant from AT&T.

1 Introduction

Recently there has been a growing need for automated tools to aid protocol designers in verifying the correctness and analyzing the performance of protocols. Protocol behavior is typically time-dependent and its correct functioning depends not only on functional requirements, but also on timing requirements [Noun 84, Shan 84]. Most of the past efforts, however, have concentrated on functional verification tools. Timing requirements of protocols have been typically ignored. Furthermore, in contrast to automated verification tools, analyses of protocol performance have been accomplished manually, (see for example [Tows 79, Bux 80]). Such analyses are based upon ad hoc and protocol-dependent techniques and thus cannot be integrated with other tools in a protocol development environment.

In this paper, the performance of a two phase locking (2PL) protocol [Bern 79] is analyzed automatically using ANALYST. ANALYST is based on a formal methodology which extracts a timing behavior of a protocol from its algebraic specification. This timing behavior can be used in analyzing the timing requirements and performance measures of protocols.

The 2PL protocol uses locking to regulate the concurrent access to shared data base items by multiple transactions. The functional correctness of such concurrency control protocols has been studied extensively [Ceri 84]. Performance analysis of these protocols, on the other hand, has just began to attract interest (see for instance [Ches 83, Morr 84]). We provide the first specification-based performance analysis of this protocol.

In section 2 we give an informal overview of the methodology underlying ANALYST. In section 3 we provide an algebraic specification of the 2PL protocol, and analyze its performance. Two timing requirements, the mean response time, and the probability of deadlock are specified and analyzed.

2 A Formal Methodology for Specification-Based Performance Analysis of Protocols

2.1 An Algebraic Specification Method

The communication behavior of a protocol may be described by expressions in a specification algebra. We assume synchronous interprocess communication in which a sender (receiver) process issuing a send (receive) event is blocked until the receiver (sender) process is ready to receive (send) it. That is, a communications block the sender and receiver processes until a successful rendezvous. Let send, receive, and rendezvous events be represented by lower case letters preceded by "!", "?", or "&", respectively.

Consider the communication behavior of a sender process S in a simple data transfer protocol. The sender sends a *message* and terminates upon receiving an *acknowledgment*, or sends the *message* again after a time-out period. This behavior can be described by a *Communication Tree* (CT) shown in Fig. 1 where nodes represent the behavior at a certain execution point and the branches represent the events. The execution starts at the root and proceeds as follows. A branch followed by a node indicates that the event labeling the branch is *sequentially* followed by the behavior represented by the node. A number of CTs connected to a node indicates that any of them can be executed *non-deterministically*. All leaves of a CT indicate *deadlock* or termination.

Such CTs can be formally defined using a universal algebra [Grat 68]. Each CT corresponds to an expression in the algebra. Let $\mathcal{E}$ denote the set of send events, receive events, and rendezvous events. Also, let the set of *identifiers* $\mathcal{I}$ refer to labels of nodes in a CT. An expression E is defined by

1. \$ (deadlock),
2. $I \in \mathcal{I}$ (identifier),
3. $e \cdot E$ (sequential composition), $e \in \mathcal{E}$,
4. $E + E$ (non-deterministic composition),

or

5. E | E (concurrent composition).

Concurrently composing two expressions produces a concurrent behavior which informally includes a rendezvous event for every pair of corresponding send and receive events, and a shuffling of other events belonging to the two expressions [Miln 80, Noun 84]. The behavior of the sender process in the data transfer protocol can be specified recursively by the following equation in S:

$$S = !message \bullet (?acknowledgment \bullet \$ + \&time\text{-}out \bullet S)$$

where *&time–out* is a rendezvous event between the sender process and a timer process. This equation is represented in the CT by the node labeled S connected to the CT whose behavior is represented by the expression on the right hand side of the equation. Given the behavior of the receiver process R in the data transfer protocol

$$R = !acknowledgment \bullet \$$$

then

$$S \mid R = S\,R$$
$$= \&acknowledgment \bullet .\$ + \&time\text{-}out \bullet (S \mid \$)$$

(We assume that concurrently composing two identifiers produces a new identifier that is a concatenation of the two former identifiers.)

In other words, the concurrent composition of expressions produces a composite expression which is expressed in terms of the sequential composition, non-deterministic composition, and deadlock operations. Any expression A in the algebra of CTs can then be represented canonically as a *sum of summands* $\sum_{i=1}^{n} a_i \bullet A_i$.

A protocol can be specified as a list of processes. Assuming that the simple data transfer protocol involves also a medium process M (whose behave includes a rendezvous event denoting message and acknowledgment loss) and a receiver process R, its specification would be given by

PROTOCOL DATA_TRANSFER : S, M, R

The concurrent behavior of the protocol can be obtained by concurrently composing the specification of its processes. During the composition, any *deadlock or unspecified reception errors* in the protocol behavior can be detected.

2.2 A Specification-Based Performance Analysis

In analyzing the performance of protocols, the specification and analysis of timing requirements and performance measures need be addressed. For example, a timing requirement for the data transfer protocol example described above, and assuming that its medium process can lose messages, would be to ensure that the probability of time-out occurring before a loss in the medium is minimal and that a time-out occurs as soon as possible after a loss [Noun 84]. An example of a performance measure is the mean roundtrip delay starting from sending a message and ending with receiving its acknowledgment at the sender.

The two aspects of protocol performance can be specified and analyzed using a timing behavior of protocol. A methodology for extracting this timing behavior from algebraic specifications of protocols augmented with the distributions of the events involved, is described in [Noun 86]. Timing behaviors of protocols are modeled as *marked point processes* [Snyd 75]. Times between occurrences of events are assumed to be exponentially distributed random variables. Probability, mean time, and variance time attributes of the timing behavior are defined as homomorphic images of expressions in the specification algebra. If the execution point is at the root of a CT representing expression B, the probability, and the mean and variance of the time duration of A (a summand of B) are denoted by $\mathbf{P}_B(A)$, $\mathbf{M}_B(A)$, and $\mathbf{V}_B(A)$, respectively. A necessary condition for these attributes to be defined is that A is a terminating behavior meaning it includes the deadlock symbol. Three theorems in appendix I define mappings from operations in the specification algebra to operations on these attributes.

Two functions: *Terminate* and *Restrict*, have been defined to be used in isolating interesting event sequences of a protocol's concurrent behavior or segments of it. Let $\mathcal{P}$ represent the power set of a set that includes all pairs (e, I), where $e \in \mathcal{E}$ and $I \in \mathcal{I}$. Informally, $Terminate[A, P]$ maps the CT corresponding to expression $\sum_{i=1}^{n} a_i \bullet A_i$ to another CT identical to the former, with the exception that for every branch labeled a_j incident upon a node labeled A_j where $(a_j, A_j) \in P$, then node A_j is labeled with a "\$' instead. This means that the new CT represents a behavior that would terminate after executing event a_j. $Restrict[A, P]$, where

$A = \sum_{i=1}^{n} a_i \cdot A_i$, maps the CT rooted at A to another CT that is identical to the former, with the exception that every branch with a labeled event a_j restricted by $(a_j, A_j) \in P$ and $A_j \neq A$, is excluded. Complete definitions of these functions can be found in [Noun 86].

Any timing requirement or performance measure of a protocol that can be specified in terms of attributes of its timing behavior can be analyzed. For example, the timing requirements necessary for the efficient operation of the simple data transfer protocol can be specified as follows. Let C denote the concurrent behavior obtained from $S \mid M \mid R$. This concurrent behavior execution consists of sequences in which time-out occurs before a loss in the medium (which would be represented by an internal event), and other sequences in which time-out occurs after a loss. The latter sequences can be isolated using *Restrict* function on C to get C_R. The timing requirement of the protocol can be then specified as *minimize* $\mathbf{M}_C(C)$ *and maximize* $\mathbf{P}_C(C_R)$. By analyzing this timing requirement an optimal setting of the time-out period is computed. The mean roundtrip delay can be specified by $\mathbf{M}_C(C)$.

3 A Two Phase Locking Protocol

In a distributed data base system, data items are distributed among several sites. User processes, at possibly different sites, execute *transactions* that are allowed to concurrently access and modify the data items. Clearly, such concurrent access has to be controlled in order to maintain a consistent state of the data base. Locking is one policy that has been used for that purpose. Eswaren, et. al., [Eswa 76] have shown that consistency is maintained by protocols using locking if transactions do not request new locks after releasing a lock (*well formed* transactions).

A two phase locking (2PL) protocol is a concurrency control protocol that uses locking [Bern 79]. In a 2PL protocol, all transactions are well-formed and each passes through a *growing phase*, *commits*, and then pursues a *shrinking phase*. In the growing phase, a transaction goes through a loop of performing some processing actions. Whenever it needs a lock, it sends a locking request to the concerned

data item, then continues processing after its request is granted. The growing phase ends when the transaction commits, i.e., all its actions are guaranteed even if the transaction later aborts (due to failure of its process, for example). In the shrinking phase, a transaction releases all acquired locks in the same order in which they were acquired and *terminates*.

A 2PL protocol ensures consistency of the data items, but it does not guarantee absence of deadlock situations. Such situation may arise between two transactions if each is waiting for a lock acquired by the other. Deadlock can be avoided if each process locks all data items required by a transaction before initiating it (*static locking*). Otherwise, a deadlock detection and recovery mechanism has to be employed to recover from deadlock situations.

In this study, we assume the following regarding the operation of the protocol:

1. *Dynamic locking*: a process locks a data item only when it is required during the growing phase.

2. *Exclusive locks*: a lock can not be shared by more than one process simultaneously; note that no distinction is drawn between read and write locks.

3. *Locking through polling*: a process that has sent a request for acquiring a lock would retry again after a waiting period to acquire it; note that no requests are assumed to be queued at a data item.

4. *Deadlock recovery via time-outs*: a process waits for a specified period for lock acquisition and upon time-out it aborts and restarts [Ceri 84]. This mechanism aims at deadlock detection. A process might, however, time-out even when no deadlock has occurred.

Although the two last assumptions have been considered by other researchers, no work has been reported on how to optimally set the time-out and polling rates. If the time-out rate is too large, then a transaction would be unnecessarily aborted and restarted thus decreasing throughput of the protocol. If it is too small then a transaction would would for a long time after a deadlock situation has occurred to abort and consequently the response time of a transaction would be degraded. Similarly, if the polling rate is too high, then the network is flooded with polling messages and assuming. Also, assuming that a lock grant arriving while a process is

sending another request does not preempt it, then if the polling rate is too high the response time will be degraded. The same effect can be also due to that a data item scheduler receiving a lock request spends item to processes it, during which a release request might arrive and its processing delayed. If the polling rate was too small then the response time would be degraded since a process waits too long before trying again to acquire a lock. Note that the use of time-outs for deadlock detection involves local decisions to restart a transaction, minimal overhead in the response time compared with other detection mechanisms which involve elaborate computations and checks of wait-for graphs [Ceri 84].

An algebraic specification of the protocol is given in section 3.1. The concurrent behavior of the protocol is computed and the space and time complexities of computing it are examined in section 3.2. Also, some interesting behaviors belonging to this concurrent behavior are specified and derived. The performance of the protocol is analyzed in section 3.3.

3.1 An Algebraic Specification

Consider a distributed data base system with M logical processes, and N distinct data items each with a scheduler process associated with it. Let M denote the set $\{i;\ i=1,...,\mathrm{M}\}$, and N denote the set $\{j;\ j=1,...,\mathrm{N}\}$. The communications between a process P_i and a data item D_j are depicted in Fig. 2. There are three ports through which they interact: a port a_{ij} for messages to acquire a new lock to the data item, a port l_{ij} for messages to grant a lock, and a port r_{ij} for messages to release a lock. The 2PL protocol is then specified as

$$\text{PROTOCOL 2PL} : P_1 , P_2 , \ldots , P_{\mathrm{M}} , D_1 , D_2 , \ldots , D_{\mathrm{N}}$$

Before introducing the detailed algebraic specifications of processes and data item schedulers, we describe simplified versions of their CTs. These CTs are illustrated in Fig. 3 and Fig. 4, respectively. In these figures, events denoting communications between a process P_i and data item scheduler D_j are described by a subscript ij; events denoting internal events in a process P_i are described by subscript i (except $\& p_{ij}$ representing process i deciding it needs to lock data item j).

A process P_i starting a new transaction, as shown in Fig. 3, might perform some

actions and then decides it needs a lock to data item j ($\&p_{ij}$). It then sends a request to it ($!a_{ij}$) and starts a time-out timer. The process is blocked until it receives a granting of its request ($?l_{ij}$) upon which it either continues processing and acquiring more locks, or decides to commit ($\&c_i$). If after a certain waiting period the locking request is not granted, the process decides to try again ($\&g_i$) and sends another request. However, if the time-out period expires ($\&t_i$), the process suspects that it is involved in a deadlock, aborts the transaction, and restarts it. When aborting or committing a transaction, a process releases all the acquired locks ($!r_{ij}$) in the same order in which they have been acquired. We assume that there is always a transaction waiting to be executed on each process; therefore, after a transaction commits and terminates a new transaction is started immediately. In addition, it is assumed that the behavior of a restarted transaction is independent from that of the previously aborted transactions.

The behavior of a data item scheduler D_i, as shown in Fig. 4, starts at a state in which it is waiting for a locking request. The first locking request it receives ($?a_{ij}$) is granted and the scheduler is locking. Subsequent locking requests while it is still locking are ignored. A grant of the first received locking request ($!l_{ij}$) is sent to the source process and the data item is locked. The data item remains locked until it receives a release request from that process ($?r_{ij}$). Release requests received from other processes (that are aborting) are ignored.

A glossary the identifiers used in the algebraic specifications and their descriptions are given in Table 1. Identifiers are associated with subscripts denoting the identity of the process or data item scheduler whose behavior they describe. In addition, identifiers of a process specification, except for the initial identifier, are associated with an ordered list of locked (and waiting to lock) data item numbers. This allows the order of acquiring locks to be remembered and thus to release them in that order in the shrinking phase. Identifiers of a data item scheduler specification are also associated with the process number that is owning a lock for the data item to distinguish between release requests when a data item is locked.

Algebraic specifications of a process and a data item scheduler are given in Fig. 5 and Fig. 6, respectively. These specifications follow the simpler corresponding CTs

in Fig. 3 and Fig. 4, respectively. One added detail in Fig. 5 is that since a process might be involved in deadlock only if it has already locked one data item, time-out is not allowed when a process is waiting for its first lock.

3.2 Concurrent Behavior

The concurrent behavior C of the specified 2PL protocol is given by

$$C = P_1 \mid P_2 \mid \ldots \mid P_M \mid D_1 \mid D_2 \mid \ldots \mid D_N \qquad (1)$$

The time complexity of obtaining the concurrent behavior of the 2PL protocol is of $O(N!^M.M^{2N}+N!^M.M^N)$, and the space complexity is of $O(N!^M\ M^N)$. A proof of how these complexities are computed is given in [Noun 86].

These explosive time and space complexities are due largely to that every process has a different identifier to describe its behavior for every possible sequence of locks acquired. Also because every data item scheduler has a different identifier for every possible process that might lock it. Consequently, generating the concurrent behavior of the 2PL protocol with large numbers of communicating processes and data items is very expensive. Subsequently in this paper, we will examine only the case of both **M** and **N** equal to 2 (unless noted otherwise). Even in this case the concurrent behavior includes 580 equations! Therefore, instead of listing the complete concurrent behavior we describe in this section some interesting sequences belonging to it.

The concurrent behavior C describes the concurrent execution of transactions on two processes that can access any of the two available data items. It includes, for example, sequences of events in which one process, waiting to acquire a lock to a data item, is blocked because the other process has already acquired that lock. It also includes other sequences of events in which transactions are executed and committed without deadlock.

The specifications of processes and data item schedulers in the 2PL protocol given in Fig. 5 and Fig. 6, respectively, are cyclic. For example, after a transaction running on a process commits another transaction is assumed to be ready and is started. Consequently, the concurrent behavior C is also cyclic describing the execution of several successive transactions on the processes in the data base.

The first behavior that we are interested in deriving from C is the terminating behavior, denoted by C_{term}, which starts at C and ends with the transaction executing on process P_1 releasing its last lock and terminating. This behavior describes the execution of one transaction from start until termination, and the effects of other concurrent transaction on it.

C_{term} can be derived as follows

$$C_{term} = Terminate[C\ ,\ \{(\&r_{11}, {}^{*}P1^{*})\ ,\ (\&r_{12}, {}^{*}P1^{*})\}] \qquad (2)$$

where '*' matches any string and is used to indicate any identifier (recall from section 2.1 that names of identifiers in the specification of a concurrent behavior are concatenations of corresponding identifiers in the concurrently composed specifications).

Two other behaviors that will be used in specifying timing requirements necessary for the efficient performance of the specified 2PL protocol, can be derived from C_{term}. The first behavior, which we refer to as C_1, is a behavior belonging to C_{term} in which the two processes are constrained such that they time-out only after the occurrence of a deadlock situation. The second behavior, which we refer to as C_2, is a behavior belonging to C_{term} in which process P_1 is constrained such that retries to acquire an awaited lock for a data item only if that data item is free.

The *Restrict* function can be used to derive C_1 as follows. Two identifiers in the concurrent behavior C_{term}, correspond to the protocol being in a deadlock state: $F_1\,1\ F_2\,2\ W_1\,12\ W_2\,21$ and $F_1\,2\ F_2\,1\ W_1\,21\ W_2\,12$. For the first identifier, process P_1 has data item D_1 locked and is waiting to acquire lock to D_2, while process P_2 has data item D_2 locked and is waiting to acquire lock to D_1. The same description applies to the second identifier with the exception that the data items are interchanged. Therefore, to compute C_1, time-out should be allowed only if the 2PL protocol is in any of these two states.

Let $\overline{P} = \{(\&t_1, F_1\ 1\ \ F_2\ 2\ \ W_1\ 12\ \ W_2\ 21)\ ,\ \ \ldots$
$(\&t_1, F_1\ 2\ \ F_2\ 1\ \ W_1\ 21\ \ W_2\ 12)$,
$(\&t_2, F_1\ 1\ \ F_2\ 2\ \ W_1\ 12\ \ W_2\ 21)\ ,$
$(\&t_2, F_1\ 2\ \ F_2\ 1\ \ W_1\ 21\ \ W_2\ 12)\}$

then $C_1 = Restrict[C_{term}, \overline{P}]$ (3)

To derive C_2, process P_1 should be allowed only to retry for an awaited lock if that lock is available. The identifiers corresponding to the cases when process P_1 is waiting to acquire the lock to data item D_1 that is free are $D_1 * W_1\ 1\ *$ and $D_1 * W_1\ 21\ *$. Similarly, the identifiers corresponding to the cases when process P_1 is waiting to acquire the lock to data item D_2 that is free are $* D_2\ W_1\ 2\ *$ and $* D_2\ W_1\ 12\ *$.

Let $\overline{P} = \{(\&C_1, D_1 * W_1\ 1\ *)\ , (\&C_1, D_1 * W_1\ 21\ *)\ ,$
$(\&C_1, * D_2\ W_1\ 2\ *)\ , (\&C_1, * D_2\ W_1\ 12\ *)\}$

then $C_2 = Restrict[C_{term}, \overline{P}]$ (4)

The last two behaviors that we are interested will be derived from the complete concurrent behavior C. These behaviors are to be used in analyzing the behavior of the protocol in deadlock situations. Consider the terminating behavior, C_{dead}, representing those behaviors of the protocol in which the protocol terminates when a deadlock occurs. Thus we can examine the deadlock behavior of the protocol without giving a chance for time-outs to resolve these deadlocks. C_{dead} can be computed from C by

$C_{dead} = Terminate[C\ , \{(\&a_{11}, F_1\ 1\ \ F_2\ 2\ \ W_1\ 12\ \ W_2\ 21)\ ,$
$(\&a_{12}, F_1\ 2\ \ F_2\ 1\ \ W_1\ 21\ \ W_2\ 12)$,
$(\&a_{21}, F_1\ 1\ \ F_2\ 2\ \ W_1\ 12\ \ W_2\ 21)\ ,$
$(\&a_{22}, F_1\ 2\ \ F_2\ 1\ \ W_1\ 21\ \ W_2\ 12)\}$ (5)

Now let us derive behavior C_3 which includes only those events sequences that lead to deadlock. C_3 can be derived from C_{dead} by constraining $\&a_{11}$ and $\&a_{12}$ such that process P_1 does not lock the two available data items and therefore there would be no possibility of deadlock. Also, $\&c_1$ should be constrained in order to avoid committing before allowing deadlock to occur. If Γ denotes the set of all identifiers in C, then C_3 is given by

$$C_3 = Restrict[C_{dead}, \{(\&a_{11}, I \in (\Gamma - D_1 * S_1\ 21)),$$
$$(\&a_{21}, I \in (\Gamma - * D_2\ S_1\ 12)),$$
$$(\&c_1, I \in \{\emptyset\})\}] \qquad (6)$$

3.3 Performance Analysis

3.3.1 Timing Model

Let λ_e denote the exponential rate of the occurrence time of event $\&e$. The rates of the events included in C are described as follows:

$\lambda_{p_{ij}}$: rate of process i accessing data item j.

λ_{t_i} : rate of time-out of process i.

λ_{g_i} : rate of polling of process i for awaited lock.

λ_{c_i} : rate of committing of process i.

$\lambda_{a_{ij}}$: rate of transmission plus propagation, and processing of locking request from process i to data item j.

$\lambda_{l_{ij}}$: rate of transmission plus propagation, and processing of granting a lock from data item j to process i.

$\lambda_{r_{ij}}$: rate of transmission plus propagation, and processing of a release request from process i to data item j.

Let the delay incurred in the transmission, propagation, and processing of a locking request, granting, or release request be denoted by $\delta_{i,j} = 1/\lambda_{a_{i,j}} = 1/\lambda_{l_{i,j}} = 1/\lambda_{r_{i,j}}$ for any i, j. We assume that any process in the data base has the same rates of events for various transactions running on it. This is clearly true for transaction-independent rates such as delay. It is also a reasonable assumption for other transaction-dependent rates assuming that transactions running on a process belong to the same *transaction class* that has the same rates.

3.3.2 Specification and Analysis of Timing Requirements

Two timing requirements are necessary for the efficient performance of the specified 2PL protocol. The first ensures that a process times-out only after a deadlock situation in which it is involved occurs and it times-out as soon as possible in order to avoid unnecessary delay. The second ensures that a process retries to acquire an awaited lock only if the lock is available and does that as soon as possible after it has become available to avoid unnecessary delay.

Optimal settings of the time-out and polling rates that satisfy these timing requirements depend on the rates of the various events involved in the global behavior of the protocol. Consequently, a process can not optimally set its time-out and polling rates using only local knowledge about the rates of its events. It has to also *know* the state of each of the data items in the case of the first timing requirement, and of each of the other processes and data items for the second. This is obviously not feasible in a distributed system.

Alternatively, we show that if a process knows the rates of events of the other processes and data item schedulers, it can use information showing the effect of these events on its performance to optimally set its time-out and polling rates. Such information can be obtained by analyzing the timing behavior of the protocol.

As discussed in section 3.1, the first timing requirement of the 2PL protocol should ensure that the transaction response time is minimized and throughput of the protocol maximized. Instead of maximizing of throughput, we consider minimizing $1 - \mathbf{P}_{C_{term}}(C_1)$ which indicates the probability of 2PL concurrent behavior in which a process times-out unnecessarily. The second timing requirement should ensure that the transaction response time and the number of unnecessary locking requests sent, are minimized. As a measure of the number of unnecessary locking requests, we consider $1 - \mathbf{P}_{C_{term}}(C_2)$. Let

$$\rho_1 = 1 - \mathbf{P}_{C_{term}}(C_1) \tag{7}$$

and

$$\rho_2 = 1 - \mathbf{P}_{C_{term}}(C_2) \tag{8}$$

The mean time, t_r, of behavior C_{term} starting a new transaction until it commits and releases all its acquired locks including restarts due to time-out is given by

$$t_r = \mathrm{M}_{C_{term}}(C_{term}) \tag{9}$$

The two timing requirements are formally specified as:

Treq1 · *Minimize* t_r *and* ρ_1.

Treq2 . *Minimize* t_r *and* ρ_2.

Consider behavior C_{term}. In a deadlock situation the only two possible events to occur are either process P_1 or process P_2 times-out (aborts), releases its locks, and restarts. This will allow the other process to acquire the awaited lock after it is released. Using rule P4 in appendix I we find that the probability of process P_1 aborting is equal to $\lambda_{t_1}/(\lambda_{t_1}+\lambda_{t_2})$. Similarly, the probability of process P_2 aborting is equal to $\lambda_{t_2}/(\lambda_{t_1}+\lambda_{t_2})$. If λ_{t_1} is set greater than λ_{t_2}, then process P_2 has higher priority in continuing without aborting, and conversely. The two processes would have the same priority if their time-out are set equal. The latter will be assumed throughout the rest of the analysis.

In order to satisfy the two timing requirements, we vary the time-out rate for the first and the polling rate for the second and find the value that minimizes the mean time and the probability terms in each. Note that optimal time-out rate is affected by variations in the setting of the polling rate. Therefore, we iterate through computing the optimal setting of one and use that to compute the optimal setting of the other until we converge.

In Fig. 7 we plot t_r versus ρ_1 for iterations 2, and 4. In Fig. 8 we plot mean time of t_r versus ρ_2 for iterations 1, 3, and 5. Note that the two goals in both timing requirements are contradictory and therefore we replace *Treq1* and *Treq2* by *minimize* $\frac{t_r}{1-\rho_1}$ and *minimize* $\frac{t_r}{1-\rho_1}$, respectively. From the figures, optimal settings of the time-out and polling rates such that *Treq1* and *Treq2* are satisfied for iterations 1 through 5 are given in Table 2. The optimal settings of the polling rate in iterations 3 and 5 are identical to two decimal places. Thus the iterations stop at 5.

3.3.3 Specification and Analysis of Probability of Deadlock

The probability of deadlock ρ_d is given by

$$\rho_d = \mathrm{P}_{C_{dead}}(C_3) \tag{10}$$

In Fig. 9 ρ_d is plotted versus the rate of committing of process P_1 for several values of the rate of committing of process P_2. As the rate of committing of a transaction class increases the shorter the transactions. That is, transactions that are less likely to need to lock all the data items available in the data base. The figure shows that the probability of deadlock increases sharply as the length of transactions increase, especially if long transactions are running on both processes.

In Fig. 10, the probability of deadlock is plotted against δ_{11} for various $\lambda_{p_{11}} = \lambda_{p_{12}}$. Increasing the access rates leads to a smaller time spent in processing actions (let it be denoted by t_{pc}). The two rates are maintained equal to analyze the effect of varying t_{pc} on probability of deadlock while holding the access ratio constant. The figure shows that as delay increases, the probability of deadlock increases and saturates for very large delays. A large delay means that a lock request sent by a process takes a long time to reach the data item during which the other process might had the chance to lock it, thus increasing the probability of deadlock. However, for a large delay that is already larger than the delay between the other process and the data items, this increase disappears. Additionally, as the processing time decreases t_{pc} increases, the probability of deadlock decreases because of the higher probability that the process decides to commit instead of needing another lock.

3.3.4 Specification and Analysis of Mean Response Time

The mean response time of a process running one transaction including restarts is given by ρ_r. In Fig. 11, ρ_r is plotted versus the commit rate λ_{c_1} for various access rates $\lambda_{p_{11}} = \lambda_{p_{12}}$. As expected, the mean response time decreases as the commit rate increases since transactions are shorter. Increasing the access rates leads to a smaller time spent in processing actions t_{pc}; this results in a lower mean response time. However, for very large access rates, ρ_r saturates. This is partly due to the increase of probability of deadlock as t_{pc} decreases since a high probability of deadlock causes transactions to abort and restart thus increasing the mean response time.

Appendix I: Mapping rules of attributes of a protocol's timing behavior

Let $F_e(t)$ and $f_e(t)$ denote the probability distribution and density function of the occurrence time of event e. Also, for a terminating expression $C = \sum_{j=1}^{m} c_j \bullet C_j$, let $CH(C)=\{c_j ; j = 1, \ldots, m\}$ and $\partial_a(C)=C_j$ if $a = c_j$, $j = 1, \ldots, m$, or otherwise undefined.

Theorem 1

P1. $\mathbf{P}_C(a \bullet A) = \mathbf{P}_C(a) \cdot \mathbf{P}_{\partial_a(C)}(A)$

P2. $\mathbf{P}_C(\sum_{i=1}^{n} a_i \bullet A_i) = \sum_{i=1}^{n} \mathbf{P}_C(a_i \bullet A_i)$

P3. $\mathbf{P}_C(a) = \int_0^\infty \prod_{e_i \in CH(C) \neq a} [1-F_{e_i}(t)]dF_a(t)$
if and only if $a \in CH[C]$

P4. $\mathbf{P}_C(\$) = 0$ if $C \neq \$$

P5. $\mathbf{P}_C(C) = 1$

Theorem 2

M1. $\mathbf{M}_C(a \bullet A) = \mathbf{M}_C(a) + \mathbf{M}_{\partial_a(C)}(A)$

M2. $\mathbf{M}_C(\sum_{i=1}^{n} a_i \bullet A_i) = \dfrac{\sum_{i=1}^{n} \mathbf{P}_C(a_i) \cdot \mathbf{M}_C(a_i \bullet A_i)}{\sum_{i=1}^{n} \mathbf{P}_C(a_i)}$

M3. $\mathbf{M}_C(a) = \int_0^\infty \prod_{e_i \in CH(C)} [1 - F_{e_i}(t)]dt$
if and only if $a \in CH[C]$

M4. $\mathbf{M}_C(\$) = 0$

Theorem 3

V1. $V_C(a \bullet A) = V_C(a) + V_{\partial_a(C)}(A)$

V2. $$V_C\left(\sum_{i=1}^{n} a_i \bullet A_i\right) = \frac{\sum_{i=1}^{n} P_C(a_i) \cdot [V_C(a_i \bullet A_i) + M_C^2(a_i \bullet A_i)]}{\sum_{i=1}^{n} P_C(a_i)} - M_C^2\left(\sum_{i=1}^{n} a_i \bullet A_i\right)$$

V3. $$V_C(a) = 2\int_0^\infty t \prod_{e_i \in CH(C)} [1-F_{e_i}(t)]\, dt - M_C^2(a)$$
if and only if $a \in CH[C]$

V4. $V_C(\$) = 0$

References

[Bern 79] P.Bernestein, D.Shipman, and W.Wong.
Formal Aspects of Serializability in Data Base Concurrency Control.
IEEE Transactions on Software Engineering SE-5:203-216, May, 1979.

[Bux 80] W.Bux, K.Kummerle, and H.Truong.
Balanced HDLC Procedures: A Performance Analysis.
IEEE Transactions on Communications COM-28(11):1889-1898, November, 1980.

[Ceri 84] S.Ceri and G.Pelagatti.
Distributed Data Bases: Principles and Systems.
McGraw-Hill Computing Science Series, 1984.

[Ches 83] A.Chesnais and E.Gelenbe.
On the Modeling of Parallel Access to Shared Data.
Communication of the ACM 26(3):196-202, March, 1983.

[Eswa 76] K.Eswaren et al.
On the Notions of Consistency and Predicate Locks in a Relational Database System.
Communications of the ACM 19(11), 1976.

[Grat 68] G.Gratzer.
Universal Algebra.
Springer-Verlag, 1968.

[Miln 80] R. Milner.
A Calculus of Communicating Systems.
Springer Verlag, 1980.

[Morr 84] R.Morris and W.Wong.
Performance Analysis of Concurrency Control Algorithms With Nonexclusive Access.
In *Proceedings of Performance'84*, pages 87-99. Elsevier Science Publishers B.V. (North Holland), 1984.

[Noun 84] N.Nounou and Y.Yemini.
Algebraic Specification-Based Performance Analysis of Communication Protocols.
In *Proceedings of the Fourth IFIP International Workshop on Protocol Specification, Testing and Verification.* North-Holland, June, 1984.

[Noun 86] N.Nounou.
Specification-Based Performance Analysis of Protocols.
PhD thesis, Columbia Univ., 1986.

[Shan 84] A.Shankar and S.Lam.
Specification and Verification of Time-Dependent Communication Protocols.
In *Proceedings of the Fourth IFIP International Workshop on Protocol Specification, Testing and Verification.* North-Holland, 1984.

[Snyd 75] D.Snyder.
Random Point Processes.
Wiley-Interscience, 1975.

[Tows 79] D.Towsley and J.Wolf.
On the Statistical Analysis of Queue Lengths and Waiting Times for Statistical Multiplexers with ARQ Retransmission Schemes.
IEEE Transactions on Communications COM-27(4):693-702, April, 1979.

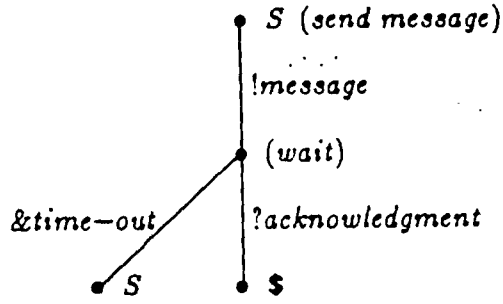


Figure 1: The CT of the sender process S

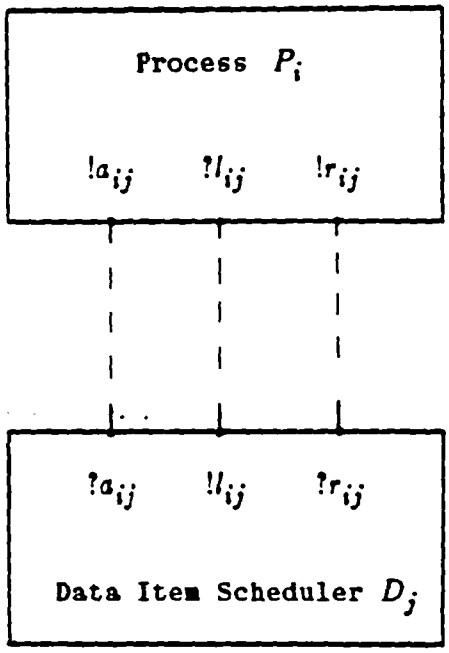
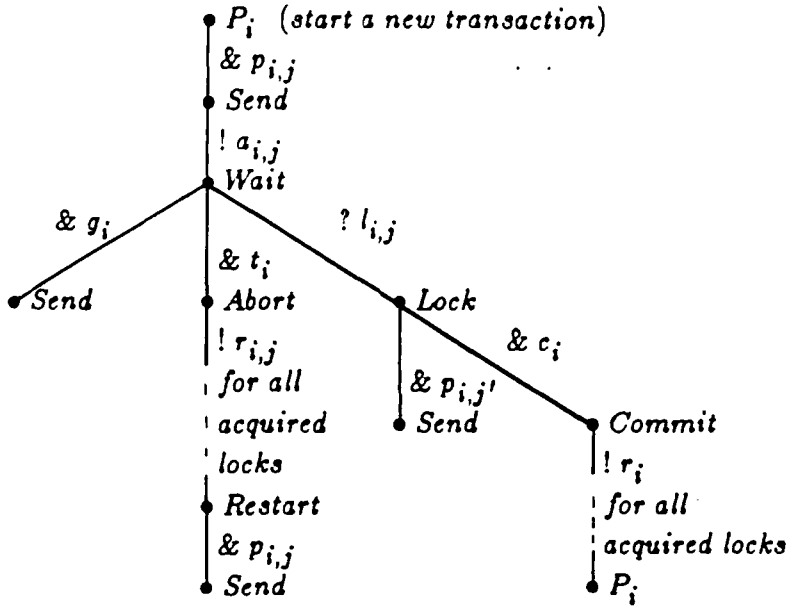
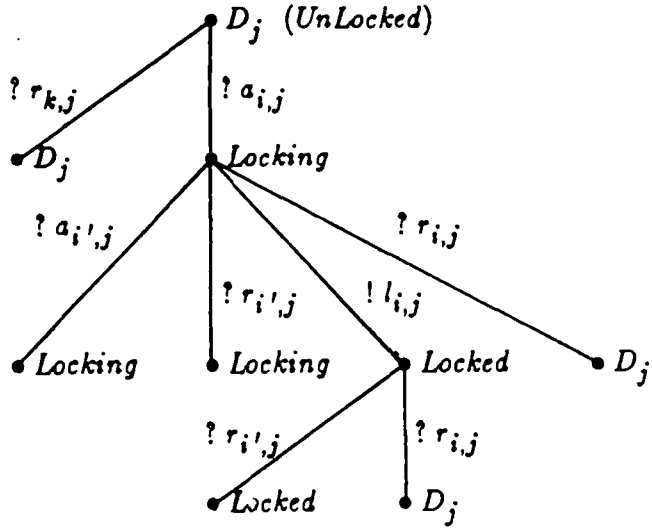


Figure 2: Communications between a process and a data item in a data base system



where $j \neq j'$; $j,j' \in \mathcal{N}$; and $i \in \mathcal{M}$

Figure 3: A simplified CT of a process in the 2PL protocol



where $i \neq i'$; $i,i',k \in \mathcal{M}$; and $j \in \mathcal{N}$

Figure 4: A simplified CT of a data item scheduler in the 2PL protocol

P_i :	process P_i is starting a new transaction.
$S_i\ j_1 j_2 \ldots j_n$:	process P_i has acquired locks for data items j_1 j_2 ... j_{n-1} and has decided to send a locking request to data item j_n.
$W_i\ j_1 j_2 \ldots j_n$:	process P_i has acquired locks for data items j_1 j_2 ... j_{n-1} and is waiting for lock of data item j_n.
$L_i\ j_1 j_2 \ldots j_n$:	process P_i has acquired locks for data items j_1 j_2 ... j_n.
$C_i\ j_1 j_2 \ldots j_n$:	process P_i has decided to commit.
$A_i\ j_1 j_2 \ldots j_n$:	process P_i is aborting.
R_i :	process P_i is restarting.
D_j :	data item D_j is unlocked.
$E_j\ i$:	data item D_j is being locked by process P_i.
$F_j\ i$:	data item D_j is locked by process P_i.

where $n \leq N$

Table 1: Glossary of identifiers used in the specification of the 2PL protocol

PROCESS P_i $\qquad 1 \leq i \leq M$

$$P_i = \sum_{j=1}^{N} \&p_{ij} \bullet S_i\ j$$

$$S_i\ j_1\ j_2 \cdots\ j_n = !a_{ij_n} \bullet W_i\ j_1\ j_2 \cdots\ j_n$$

$$W_i\ j = ?l_{ij_n} \bullet L_i\ j + \&g_i \bullet S_i\ j$$

$$W_i\ j_1\ j_2 \cdots\ j_n = ?l_{ij_n} \bullet L_i\ j_1\ j_2 \cdots\ j_n + \&g_i \bullet S_i\ j_1\ j_2 \cdots\ j_n + \&t_i \bullet A_i\ j_1\ j_2 \cdots\ j_n$$

$$L_i\ j_1\ j_2 \cdots\ j_n = \sum_{k \in N - \{j_1, \ldots, j_n\}} \&p_{ik} \bullet S_i\ j_1\ j_2 \cdots\ j_n\ k + \&c_i \bullet C_i\ j_1\ j_2 \cdots\ j_n$$

$$C_i\ j_1\ j_2 \cdots\ j_n = !r_{ij_1} \bullet C_i\ j_2 \cdots\ j_n$$

$$C_i\ j = !r_{ij} \bullet P_i$$

$$A_i\ j_1\ j_2 \cdots\ j_n = !r_{ij_1} \bullet A_i\ j_2 \cdots\ j_n$$

$$A_i\ j = !r_{ij} \bullet R_i$$

$$R_i = \sum_{j=1}^{N} \&p_{ij} \bullet S_i\ j$$

END

Figure 5: An algebraic specification of process P_i

PROCESS D_j $\qquad 1 \leq j \leq N$

$$D_j = \sum_{i=1}^{M} ?a_{ij} \cdot E_j\ i + \sum_{i=1}^{M} ?r_{ij} \cdot D_j$$

$$E_j\ i = \sum_{i=1}^{M} ?a_{ij} \cdot E_j\ i + \sum_{k \in M-\{i\}} ?r_{kj} \cdot E_j\ i + !l_{ij} \cdot F_i\ j + ?r_{ij} \cdot D_j$$

$$F_j\ i = \sum_{k \in M-\{i\}}^{M} ?a_{ik} \cdot F_j\ i + \sum_{k \in M-\{i\}} ?r_{kj} \cdot F_j\ i + ?r_{ij} \cdot D_j$$

END

Figure 6: An algebraic specification of a data item scheduler D_j

Iteration Number	Rate	Optimal Setting
1	λ_{g_1}	10.5
2	λ_{t_1}	5.25
3	λ_{g_1}	8.2
4	λ_{t_1}	5.0
5	λ_{g_1}	8.2

Table 2: Optimal settings (in occurrences/sec) of the time-out and polling rates for various iterations

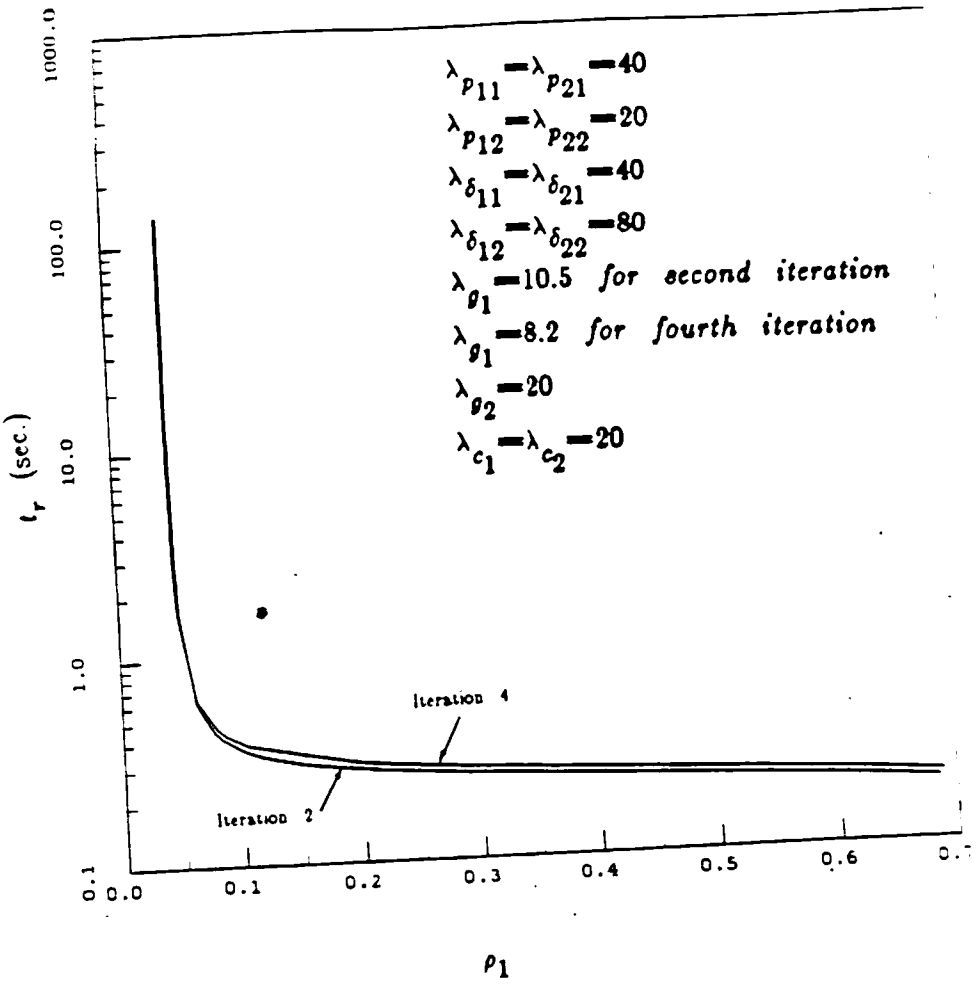


Figure 7: Transaction mean response time t_r versus probability of behavior involving unnecessary time-outs ρ_1.

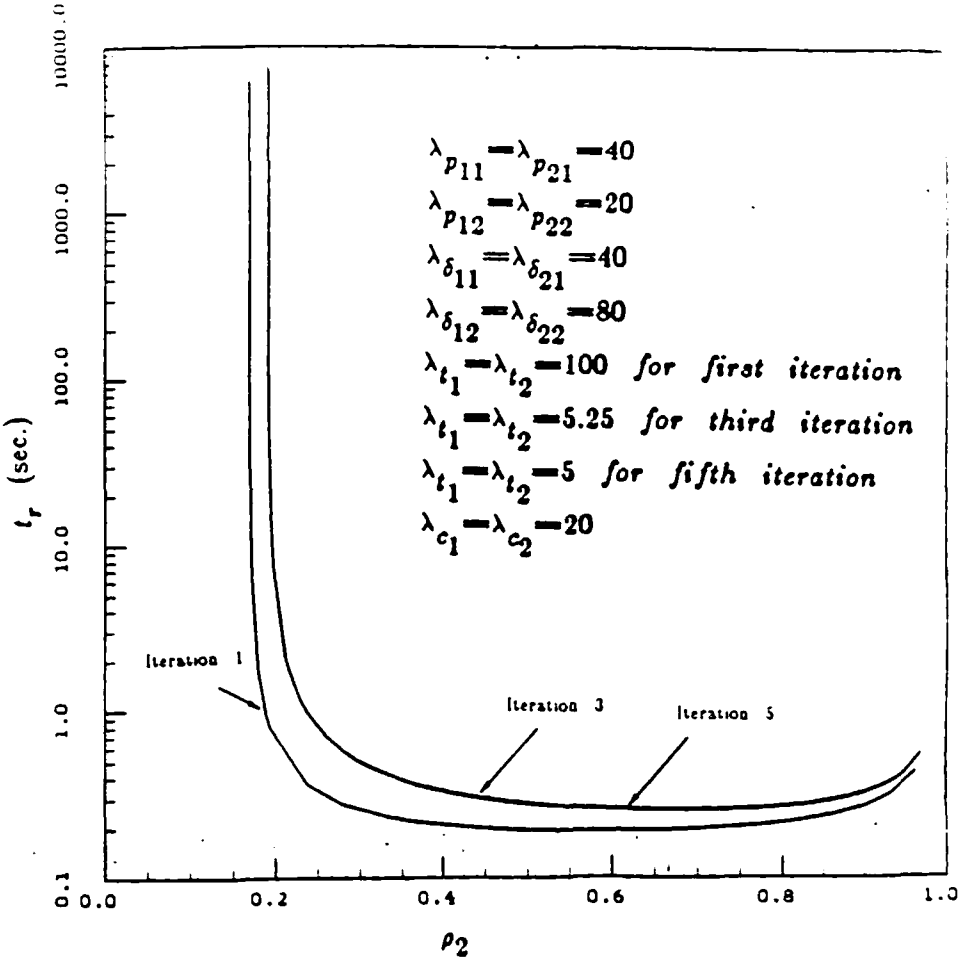


Figure 8: Transaction mean response time t_r versus probability of behavior involving unnecessary transmissions of locking requests ρ_2.

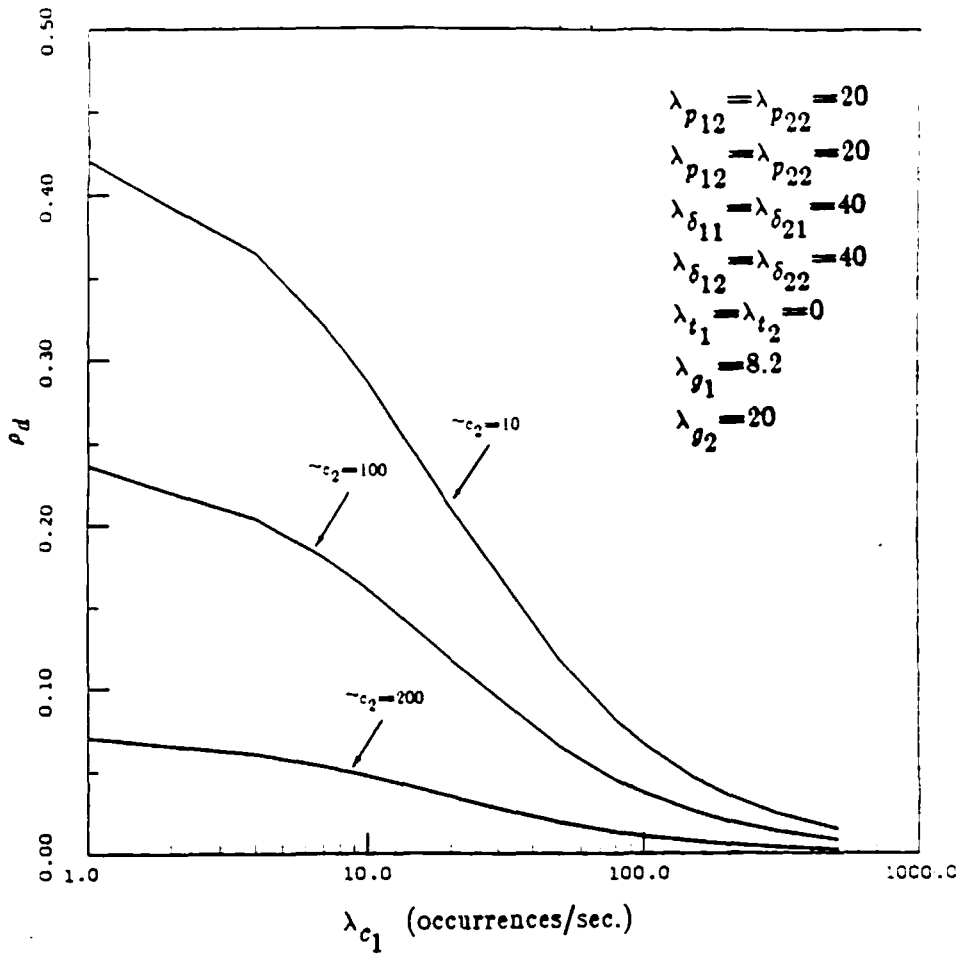


Figure 9: Probability of deadlock ρ_d versus rate of committing λ_{c_1} for various λ_{c_2}.

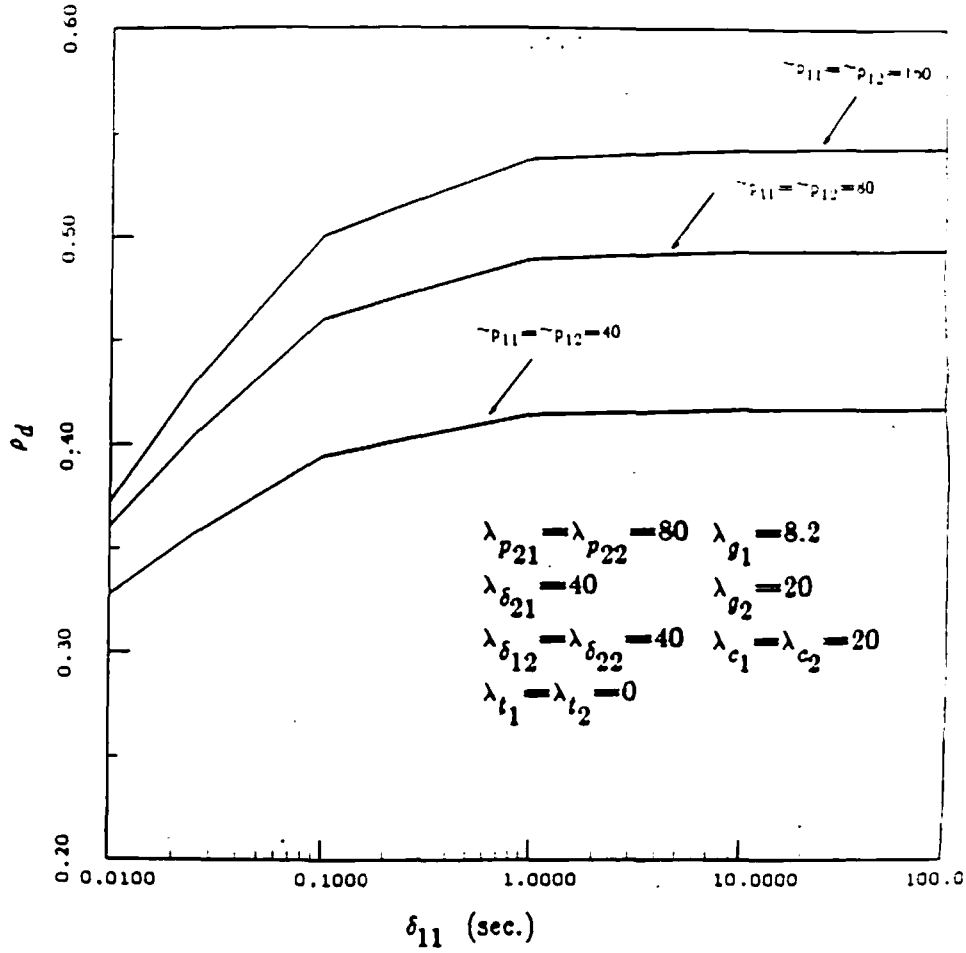


Figure 10: Probability of deadlock ρ_d versus delay δ_{11} for various $\lambda_{p_{11}}=\lambda_{p_{11}}$.

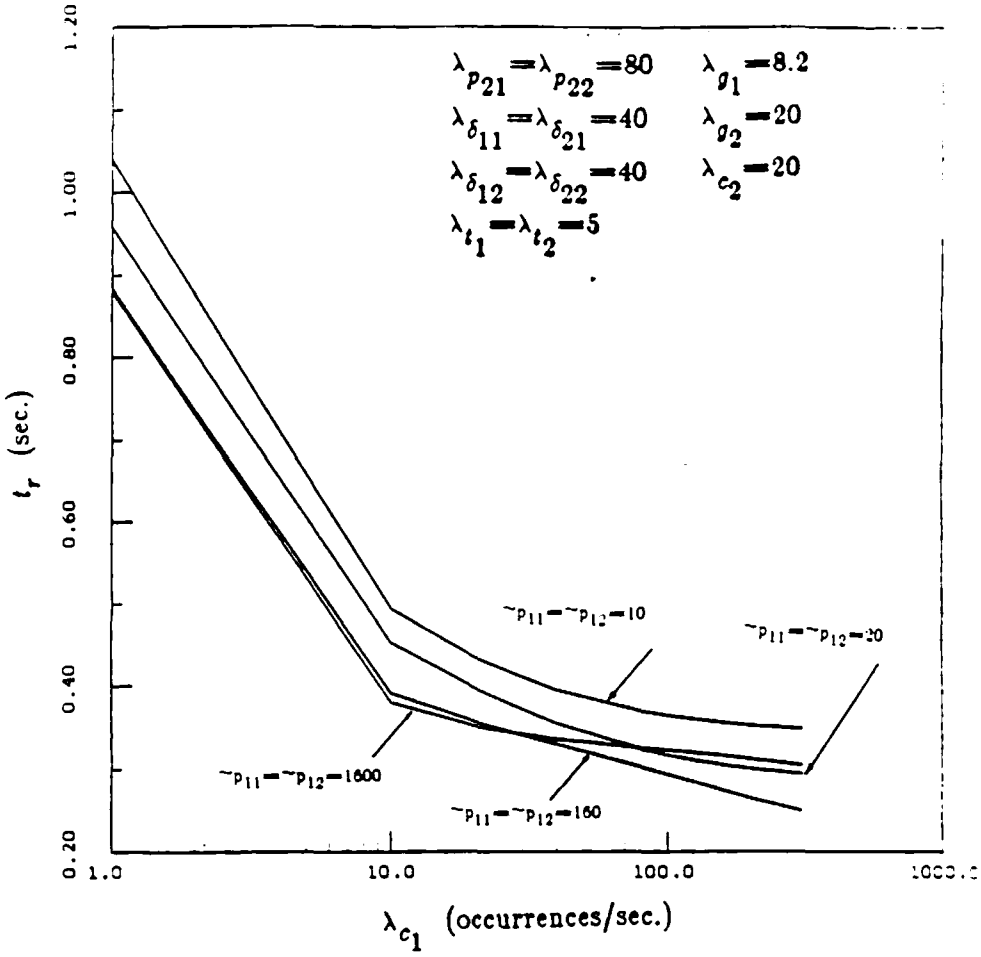


Figure 11: Transaction mean response time t_r versus commit rate λ_{c_1} for various $\lambda_{p_{11}}=\lambda_{p_{12}}$.

Table of Contents

List of Figures

List of Tables