On the Analytical Modeling of Database Concurrency Control

PHILIP S. YU, DANIEL M. DIAS, AND STEPHEN S. LAVENBERG

IBM Research Division, T. J. Watson Research Center, Yorktown Heights, New York

Abstract. The Concurrency Control (CC) scheme employed can profoundly affect the performance of transaction-processing systems. In this paper, a simple unified approximate analysis methodology to model the effect on system performance of data contention under different CC schemes and for different system structures is developed. This paper concentrates on modeling data contention and then, as others have done in other papers, the solutions of the data contention model are coupled with a standard hardware resource contention model through an iteration. The methodology goes beyond previously published methods for analyzing CC schemes in terms of the generality of CC schemes and system structures that are handled. The methodology is applied to analyze the performance of centralized transaction processing systems using various optimistic- and pessimistic-type CC schemes and for both fixed-length and variable-length transactions. The accuracy of the analysis is demonstrated by comparison with simulations. It is also shown how the methodology can be applied to analyze the performance of distributed transaction-processing systems with replicated data.

Categories and Subject Descriptors: C.4 [**Computer Systems Organization**]: Performance of Systems—*modeling techniques*: H.2.4 [**Database Management**]: Systems—*transaction processing*

General Terms: Performance, Theory

Additional Key Words and Phrases: Concurrency control, data contention, distributed systems, locking, optimistic concurrency control, resource contention

1. *Introduction*

Concurrency Control (CC) can critically affect the performance of transaction processing systems. This is particularly so as the demand for transaction throughput increases, leading to greater data contention. Projections have been made for a requirement on the order of 1000 transactions per second in the near future. With the advent of VLSI technology, coupling multiple small processors to support high transaction rates has been pursued by various vendors. However, a smaller processor does have longer execution time that translates to a longer holding time on accessed data, and coupling also introduces additional coordination overhead. These factors make the coupled system more susceptible to data contention [14]. There has also been consider-

Authors' address: IBM Research Division, T. J. Watson Research Center, Yorktown Heights, NY 10598.

able interest in geographically distributed transaction processing systems, in which the databases are distributed among regional systems [20]. The access of remote data is expensive in terms of the communications overhead and delays involved, and thus has a negative impact on the level of data contention [15].

There have been numerous analytical studies of the performance of CC schemes. A survey of the analysis of locking in centralized databases can be found in [33] and [34], while Sevcik [31] examines early analytical work on the performance of different CC schemes in distributed databases. More recent work includes [5–9], [12], [22], [23], [28], [30], [32], [35], [36], [39], [40], [43–47]. Although there have been numerous simulation studies of concurrency control performance, we do not mention them here since the focus of this paper is on analytical methodology.

In this paper, we develop a unified analytical methodology to approximate the mean transaction response time in transaction processing systems for a variety of CC schemes. This methodology goes beyond previously published methods for analyzing CC schemes in terms of its generality. The intent is to provide basic approximations that can be systematically applied to a variety of cases. Our analysis is based on probabilistic assumptions that are explicitly stated. Based on these assumptions, we obtain simple approximate expressions for the probability of data conflict for both locking and optimistic CC and for schemes that combine both methods. For locking, we also obtain a simple approximate expression for the mean lock waiting time. When considering mean response time the effects of both data contention and hardware resource contention must be accounted for. As has been done before (e.g., [38] and [45]) we capture both effects by iterating between the solutions of the CC model and a (conventional) queuing model of hardware contention. We show that our methodology is applicable to both fixed- and variable-length transactions, multiple transaction types, nonuniform data access, and multiple lock modes (e.g., shared and exclusive). We present our methodology in the context of an open system model, but also show how it can be applied to a closed model with a fixed multiprogramming level. Simulations are used to demonstrate the accuracy of the analytical results in a variety of cases. Our main focus is on centralized systems, but we also show how the methodology can be applied to distributed systems.

The methodology in this paper is derived and extended from a number of specific studies we have done (e.g., [7–10] and [42–47]). In addition to the comparisons with simulation results presented here, comparisons with simulations are presented in those papers for similar approximations.

In Section 2, we discuss the general methodology. We apply this methodology in Section 3 to analyze various CC schemes for centralized systems. For optimistic schemes, the effect of changed buffer hit probability when a transaction is rerun is handled in the analysis. For locking, we obtain a simple approximate expression for the mean lock waiting time. Comparisons with simulation results are given. In Section 4, we show how the methodology can be applied to distributed systems with replicated data. Concluding remarks appear in Section 5.

2. *General Methodology*

In an environment with no data contention, the transaction response time is determined by the queuing and processing delay in accessing hardware re-

sources such as CPU, I/O, etc. In the presence of data contention, the transaction response time further depends upon the occurrence of data conflict in accessing the database. The probability of data conflict for any transaction depends not only on the CC scheme itself but also on the transaction response time, which in turn depends on the conflict probability. For example, when locking is used, if the lock contention probability increases, the transaction response time increases due to additional lock waits. In turn, longer transaction response time leads to a longer lock holding time, and hence to a higher lock contention probability.[1] Similarly, when an optimistic CC scheme is used if the transaction abort rate increases, there is a concomitant increase in the CPU utilization, causing longer response and data-holding times, and therefore a higher probability of abort. We model hardware resource access times and the effect of CC separately, and then we solve the models simultaneously using an iteration to estimate the mean transaction response time. (As mentioned in the introduction, the iterative approach is not new.) We assume an open model with Poisson transaction arrivals, but in addition show in Section 3.4 how our analysis can be extended to a closed model. The hardware resource contention can be modeled by conventional queuing models with CPU servers and disk servers. Since this is rather straightforward, we only briefly outline it for the case of analyzing pure optimistic CC (Section 3.1.1.2). A model for predicting buffer hit probability can also be incorporated to provide a more accurate estimate of the number of IOs [11]. In this paper, we concentrate on how to estimate the data conflict probability and mean lock waiting time.

We first describe the transaction model. The transaction model consists of $n_L + 2$ states, where n_L is the random number of granules accessed by the transaction. (We use the term, *granule*, to refer to the unit of data to which concurrency control is applied.) The transaction has an initial setup phase (including program fetch, and message processing), state 0. Following the initial setup, a transaction progresses to states $1, 2, \ldots, n_L$, in that order. This is the execution phase. At the start of each state, i, the transaction begins to access a new granule and moves to state $i + 1$ when the next new granule access begins. At the end of state n_L, if successful, the transaction enters into the commit phase at state $n_L + 1$.

A *conflict* is defined to be an event in which a transaction accesses a data granule that is currently accessed or in use by another transaction in an incompatible mode. The result of a conflict is either a transaction wait or transaction abort. Let r be the random variable denoting the transaction response time; r_{INPL}, the execution time in state 0 (the initial setup phase); r_{E_i}, the sum of the execution times in states $1, \ldots, n_L$, excluding lock waiting times, for the ith run of a transaction; $t_{\text{Backoff}_{i+1}}$, the time from the ith abort of a transaction until its $(i + 1)$st run is started; and t_{Commit}, the commit time to reflect the updates into the database. Note that these random variables include processing and I/O times as well as the queuing times for these hardware

[1] In [32] an exact combinatorial model for the analysis of the probability of conflict between two transactions with nonuniform access using static locking (i.e., all locks acquired at the beginning of a transaction) is examined. This method can generalize to calculate the probability of conflict among a set of transactions assuming that the number of transactions and their sizes are known. However, as described above, the probability of conflict affects the response time, which in turn affects number of transactions in the system, and therefore the method in [32] does not estimate system performance measures such as throughput and response time.

resources. Let n_A be the random variable denoting the number of runs of a transaction, n_{W_i} be the number of lock waits during the ith run of a transaction, and $r_{W_{ij}}$ be the waiting time for the jth lock contention during the ith run. The transaction response time r can be expressed as,

$$r = r_{\text{INPL}} + r_{E_1} + \sum_{j=1}^{n_{W_1}} r_{W_{1j}} + \sum_{i=2}^{n_A} \left(t_{\text{Backoff}_i} + r_{E_i} + \sum_{j=1}^{n_{W_i}} r_{W_{ij}} \right) + t_{\text{Commit}}. \quad (2.1)$$

In this paper, we use lowercase letters to represent random variables and uppercase letters to represent the means of the corresponding random variables. We are interested in approximating the mean response time, R. When we analyze locking, we assume that deadlocks do not occur. Other studies, as well as our simulations, have shown that the probability of deadlock is small compared to the probability of lock contention and that deadlock can be ignored when analyzing system performance (e.g., [21], [35], and [40]). Thus, for locking, there are no reruns and $n_A = 1$. If, in addition, we assume that the lock waiting times for a transaction are independent with the same mean R_W, and that the number of lock waits for a transaction is a stopping time for its sequence of lock waits, then it follows from Eq. (2.1) that

$$R = R_{\text{INPL}} + R_E + N_W R_W + T_{\text{Commit}}$$

where we have dropped the subscript 1 from R_E and N_W. If we further assume the lock contention events for a transaction are independent with the same probability of occurrence P_W and are independent of the number of granules accessed, then $N_W = P_W N_L$ and

$$R = R_{\text{INPL}} + R_E + N_L P_W R_W + T_{\text{Commit}}. \quad (2.2)$$

For optimistic CC, there are no lock waits so that from Eq. (2.1)

$$r = r_{\text{INPL}} + r_{E_1} + \sum_{i=2}^{n_A} \left(t_{\text{Backoff}_i} + r_{E_i} \right) + t_{\text{Commit}}.$$

We allow a rerun of a transaction to have different mean execution time and abort probability from the first run of the transaction. (The reason for this is discussed in Section 3.1.) If we assume that (i) the backoff times are independent with common mean T_{Backoff}, (ii) the execution times for each rerun are independent with common mean R'_E, (iii) the number of runs, n_A, is a stopping time for the sequence $\{t_{\text{Backoff}_i} + r_{E_i}\}$ where $t_{\text{Backoff}_1} = 0$, and (iv) the occurrence of aborts of all runs are independent and the probability of occurrence for the first run is P_A and for any subsequent run is P'_A, then

$$R = R_{\text{INPL}} + R_E + \frac{P_A}{(1 - P'_A)} + (T_{\text{Backoff}} + R'_E) + T_{\text{Commit}}, \quad (2.3)$$

where we have omitted the subscript 1 from R_E. In Eqs. (2.2) and (2.3), R_{INPL}, R_E, R'_E, T_{Backoff}, and T_{Commit} can be approximated from the hardware queuing model. The remaining quantities to be approximated are P_W, the probability of lock contention, R_W, the mean lock waiting time and the abort probabilities, P_A and P'_A. We next present our methodology for approximating the lock contention and abort probabilities. We postpone approximating R_W until Section 3.2.

2.1. CONFLICT ANALYSIS. We make the following assumption, which is used in obtaining all the analytical results: All granule access requests are independent random variables. Although it is reasonable to assume that granule access requests from different transactions are independent, independence cannot hold within a transaction if a transaction's granule accesses are distinct. However, we use this assumption as an approximation for the granule accesses from the same transaction. If the probability of accessing any particular granule is small, e.g., when the number of granules is large and the access distribution is uniform, this approximation should be very accurate. A similar approximation is also made in [33], [35], and [36].

We make the following additional assumptions and then show how they can be relaxed at the end of this section:

—There is only one transaction type;
—The granule access distribution is uniform over the set of granules;
—All accesses to granules are exclusive or update.

Let λ be the transaction arrival rate, N_L be the mean number of granule accesses by each transaction, and L be the number of granules in the database. Let T_{H} denote the mean holding time of a granule, that is, the period of time from when the granule is first accessed until the end of the transaction. In all the analyses that follow, we assume that the system is stable and ergodic so that quantities like the mean granule holding time, the lock contention probability, and the mean transaction response time exist and are finite and defined to be either long-run averages or steady-state quantities.

Consider a generic locking scheme. (We mention again that we assume that deadlocks do not occur.) Before a granule can be accessed, a transaction needs to acquire a lock on that granule. A lock request can be made right before each granule is accessed as in dynamic locking, or all lock requests can be made at the beginning of a transaction as in static locking. Each time a lock request is made the corresponding entry in the lock table is examined. If no other transactions hold an incompatible lock on that entry, the granule is locked and the access is granted. The granule is locked until it is released by the transaction. If the granule has already been locked in an incompatible mode, lock contention occurs. (Note that for now we are considering only exclusive granule access so that all accesses to the same granule are incompatible.) Due to the assumption that deadlocks do not occur and the assumption of uniform granule access distribution, the arrival rate of lock requests for a particular granule is $\lambda N_L/L$.

PROPOSITION 2.1. *Assume that the lock request times form a Poisson process and that lock contention events for a transaction are independent. Then the probability of contention on any lock request, P_W, is given by*

$$P_W = \frac{\lambda N_L T_H}{L}, \tag{2.4}$$

and the probability, P_{CONT}, that a transaction encounters lock contention is upper bounded as follows:

$$P_{CONT} \leq 1 - \left(1 - \frac{\lambda N_L T_H}{L}\right)^{N_L}. \tag{2.5}$$

PROOF. The arrival rate of lock requests for a granule is $\lambda N_L/L$. Given a mean lock holding time of T_H, the lock utilization is $\lambda N_L T_H/L$. Using the assumption that the lock request times are a Poisson process, the probability of contention on a lock request equals the lock utilization, $\lambda N_L T_H/L$, which establishes (2.4).

Let n_j be the number of granules accessed by the jth transaction to arrive at the system. Then using the assumption of independent lock contention events,

$$P\{\textit{trans. } j \textit{ encounters lock contention}|n_j\} = 1 - \left(1 - \frac{\lambda N_L T_H}{L}\right)^{n_j}.$$

Applying Jensen's inequality to the above equation yields,

$$P_{CONT} = E\left[1 - \left(1 - \frac{\lambda N_L T_H}{L}\right)^{n_j}\right] \leq 1 - \left(1 - \frac{\lambda N_L T_H}{L}\right)^{E[n_j]}$$

$$= 1 - \left(1 - \frac{\lambda N_L T_H}{L}\right)^{N_L}. \qquad \text{Q.E.D.}$$

We use the upper bound in Eq. (2.5) as an approximation to P_{CONT}, that is,

$$P_{CONT} \simeq 1 - \left(1 - \frac{\lambda N_L T_H}{L}\right)^{N_L}. \tag{2.6}$$

An $O(1/L)$ approximation to Eq. (2.6) is given by

$$P_{CONT} \simeq \frac{\lambda N_L^2 T_H}{L}. \tag{2.7}$$

We next consider a generic Optimistic CC (OCC) scheme. Transactions access granules as they progress. At the end of execution, if all granules accessed are the up-to-date versions, the transaction will commit and reflect the updated values into the database.[2] In this generic OCC scheme, the commit is assumed to be instantaneous. The effect of nonzero commit time is addressed in later sections when specific certification schemes are considered. At the end of the commit phase, for each granule updated, any transaction accessing the granule is notified that the granule is invalid. Transactions accessing invalid granules are aborted at commit time (referred to as *pure OCC*). (Later, we consider the case where transactions accessing invalid granules are aborted immediately, referred to as *broadcast OCC*.) The rate of invalidation to a particular granule is equal to the transaction throughput multiplied by the mean number of granules updated for each transaction and divided by the database size, that is, it is given by $\lambda N_L/L$.

PROPOSITION 2.2. *Assume that the invalidation times for a given granule form a Poisson process and that all such processes (to different granules) are indepen-*

[2] We assume that during the course of a transaction any updates are kept in a private buffer, and are only made visible to other transactions after a successful commit, as in [2] and [25].

dent. Then the probability, P_A, that a transaction is aborted when it first tries to commit is upper bounded as follows:

$$P_A \leq 1 - exp\left(-\frac{\lambda N_L^2 T_H}{L}\right), \tag{2.8}$$

where in Eq. (2.8) T_H *is the mean granule holding time for transactions during their first run only.*

PROOF. Let t_{ij} be the holding time for the ith granule accessed by the jth transaction during its first run. Let n_j be the number of granules accessed by the jth transaction. It follows from the Poisson assumption that

$$P\{\text{trans. } j \text{ is invalidated due to the } i\text{th granule it accessed}|t_{ij}\} = 1 - \exp\left(-\frac{\lambda N_L t_{ij}}{L}\right).$$

It follows from the independence assumption that

$$P\{\text{trans. } j \text{ invalidated}|n_j, t_{1j}, t_{2j}, \ldots, t_{n_j j}\} = 1 - \prod_{i=1}^{n_j} \exp\left(-\frac{\lambda N_L t_{ij}}{L}\right) = 1 - \exp\left(-\frac{\lambda N_L}{L}\sum_{i=1}^{n_j} t_{ij}\right).$$

(A similar expression was also obtained in [30, Eq. (27)], but the rest of our analysis differs from theirs.) Applying Jensen's inequality we have,

$$P\{\text{trans. } j \text{ invalidated}\} = 1 - E\left[\exp\left(-\frac{\lambda N_L}{L}\sum_{i=1}^{n_j} t_{ij}\right)\right] \leq 1 - \exp\left(-\frac{\lambda N_L}{L}E\left[\sum_{i=1}^{n_j} t_{ij}\right]\right).$$

Then[3]

$$P_A = \lim_{J\to\infty}\frac{1}{J}\sum_{j=1}^{J} P\{\text{trans. } j \text{ invalidated}\} \leq 1 - \lim_{J\to\infty}\frac{1}{J}\sum_{j=1}^{J}\exp\left(-\frac{\lambda N_L}{L}E\left[\sum_{i=1}^{n_j} t_{ij}\right]\right).$$

Applying Jensen's inequality to the averaging over j

$$P_A \leq 1 - \lim_{J\to\infty}\exp\left(-\frac{\lambda N_L}{L}\frac{1}{J}\sum_{j=1}^{J}E\left[\sum_{i=1}^{n_j} t_{ij}\right]\right) = 1 - \exp\left(-\frac{\lambda N_L}{L}\lim_{J\to\infty}\frac{1}{J}\sum_{j=1}^{J}E\left[\sum_{i=1}^{n_j} t_{ij}\right]\right).$$

[3] The equality in the equation follows from ergodicity and the Dominated Convergence Theorem.

Now, the average holding time, T_H, for a granule during the first run of a transaction can be expressed as,[4]

$$\begin{aligned} T_H &= \lim_{J\to\infty} \frac{1}{n_1 + \cdots + n_J} \sum_{j=1}^{J} \sum_{i=1}^{n_j} t_{ij} \\ &= \lim_{J\to\infty} \frac{(1/n_1 + \cdots + n_J)}{J} \frac{1}{J} \sum_{j=1}^{J} \sum_{i=1}^{n_j} t_{ij} \\ &= \frac{1}{N_L} \lim_{J\to\infty} \frac{1}{J} \sum_{j=1}^{J} \sum_{i=1}^{n_j} t_{ij} \\ &= \frac{1}{N_L} \lim_{J\to\infty} \frac{1}{J} \sum_{j=1}^{J} E\left[\sum_{i=1}^{n_j} t_{ij}\right]. \end{aligned} \tag{2.9}$$

Using this in the above inequality for P_A,

$$P_A \le 1 - \exp\left(-\frac{\lambda N_L^2 T_H}{L}\right). \qquad \text{Q.E.D.}$$

Note that Eq. (2.9) needs not hold for rerun transactions since the mean length of rerun transactions needs not equal that of first-run transactions except if all transactions have the same fixed length. We use the upper bound in Eq. (2.9) as an approximation to P_A, that is,

$$P_A \simeq 1 - \exp\left(-\frac{\lambda N_L^2 T_H}{L}\right). \tag{2.10}$$

An $O(1/L)$ approximation to Eq. (2.10) is

$$P_A \simeq \frac{\lambda N_L^2 T_H}{L}. \tag{2.11}$$

Next we consider the case of multiple transaction types. Let λ_i, N_{L_i}, and T_{H_i} be the entities corresponding to λ, N_L, and T_H for type i transactions. In the following, we present approximations for the probability of conflict with an arbitrary type of transaction and between transaction types. These approximations are derived in a similar manner to that used to derive the approximations in Eqs. (2.7) and (2.11), and so we omit the derivations.

First note that, similar to Eq. (2.4), the lock contention probability, P_W^i, of an arbitrary lock request with type i transactions, and the overall lock contention

[4] In Eq. (2.9), general conditions under which the last equality, which involves an interchange of the limit and expectation operators, occurs are beyond the scope of this paper. However, if $n_j \le N_{max}$ for all j, where N_{max} is a finite constant, then, $\sum_{i=1}^{n_j} t_{ij} < N_{max} r_j$ where r_j is the response time of transaction j. From ergodicity $\lim_{J\to\infty}(1/J)\sum_{j=1}^{J} r_j = R_T$. If $R_T < \infty$ and, in addition, $E[r_j] < \infty$ for all j, it can be shown that the last equality in Eq. (2.9) follows from the Dominated Convergence theorem.

probability, P_W, of a lock request are

$$P_W^i = \frac{\lambda_i N_{L_i} T_{\mathrm{H}_i}}{L};$$

$$P_W = \sum_i \frac{\lambda_i N_{L_i} T_{\mathrm{H}_i}}{L}.$$

A type j transaction contention probability, $P_{\mathrm{CONT}}^{j,i}$, with type i transactions, and the overall contention probability, P_{CONT}^{j}, of type j transactions can be approximated as,

$$P_{\mathrm{CONT}}^{j,i} \simeq \frac{N_{L_j} \lambda_i N_{L_i} T_{\mathrm{H}_i}}{L};$$

$$P_{\mathrm{CONT}}^{j} \simeq \frac{N_{L_j} \Sigma_i \lambda_i N_{L_i} T_{\mathrm{H}_i}}{L}. \tag{2.12}$$

Under OCC, the invalidation probability, $P_A^{j,i}$, of a type j transaction by type i transactions can be approximated as

$$P_A^{j,i} \simeq \frac{N_{L_j} T_{\mathrm{H}_j} \lambda_i N_{L_i}}{L},$$

and hence the overall invalidation probability, P_A^j, for a type j transaction can be approximated as

$$P_A^j \simeq \frac{N_{L_j} T_{\mathrm{H}_j} \Sigma_i \lambda_i N_{L_i}}{L}. \tag{2.13}$$

Assuming that the granule updates/invalidations from different transaction types are independent we can also use the following approximation,

$$P_A^j = 1 - \prod_i \left(1 - P_A^{j,i}\right) \simeq 1 - \prod_i \left(1 - \frac{\lambda_i N_{L_i} N_{L_j} T_{\mathrm{H}_j}}{L}\right).$$

Returning to the case of a single transaction type, we next consider the case of nonuniform accesses to the database. Assume that the database consists of multiple (I) sets of granules with L_i granules in set i. Assume that each transaction makes n_{L_i} (with mean N_{L_i}) granule requests to set i (with $N_L = \Sigma_{i=1}^{I} N_{L_i}$) and that the accesses to a set are uniform over the set. Let T_{H_i} be the mean granule holding time for set i. The lock utilization for set i is $\lambda N_{L_i} T_{H_i} | L_i$. Assume Poisson arrivals of lock requests and that lock contention events for a transaction are independent. Then,

$$P\{\text{transaction encounters lock contention} | n_i \text{ requests to set } i, i = 1, \ldots, I\}$$

$$= 1 - \prod_{i=1}^{I} \left(1 - \frac{\lambda N_{L_i} T_{\mathrm{H}_i}}{L_i}\right)^{n_i}.$$

A similar expression was obtained in the proof of Proposition 2.1. In the same way that the approximation in Eq. (2.7) was shown to follow from the

expression, it follows that

$$P_{\text{CONT}} \simeq \sum_{i=1}^{I} \frac{\lambda N_{L_i}^2 T_{\text{H}_i}}{L_i}. \tag{2.14}$$

Assuming that each lock request independently references set i with probability N_{L_i}/N_L, then the mean lock holding times are the same for each set i, that is, $T_{H_i} = T_H$ $1 \le i \le I$. Then Eq. (2.14) can be expressed as $P_{\text{CONT}} \simeq \lambda N_L^2 T_H / \hat{L}$ where $\hat{L} = N_L^2 / \sum_{i=1}^{I} (N_{L_i}^2 / L_i)$. Comparing this with Eq. (2.7), $\hat{L}$ can be considered to be the effective database size. For example, if a fraction α of the accesses are to a fraction β of the granules, then $\hat{L} = L/(\alpha^2/\beta + (1-\alpha)^2/(1-\beta))$, which is less than L if $\alpha \neq \beta$. Thus, hot sets reduce the effective database size. Under OCC in this environment, a similar development to that for a single set of granules gives,

$$P_A \simeq \sum_{i=1}^{I} \frac{\lambda N_{L_i}^2 T_{\text{H}_i}}{L_i}. \tag{2.15}$$

Finally, we illustrate how the case of different access modes can be handled by considering the case of exclusive and shared modes. Assume that each data access has probability P_{EXCL} of being in exclusive mode and that the mode of each access is independent. Again we consider the case of one transaction type and uniform accesses to the database. Let n_j be the number of granules accessed by the jth transaction, and let $n_{j,\text{EXCL}}$ be the number of granules accessed in exclusive mode by the jth transaction. First, consider the generic locking case. Then,

$$P\{\text{trans. } j \text{ encounters lock contention} | n_j, n_{j,\text{EXCL}}\}$$
$$= 1 - \left(1 - \frac{\lambda N_L T_{\text{H}}}{L}\right)^{n_{j,\text{EXCL}}} \left(1 - \frac{\lambda N_L P_{\text{EXCL}} T_{\text{H}}}{L}\right)^{(n_j - n_{j,\text{EXCL}})}.$$

Again using a similar approach to that in the proof of Proposition 2.1, we get

$$P_{\text{CONT}} \le 1 - \left(1 - \frac{\lambda N_L T_{\text{H}}}{L}\right)^{N_L P_{\text{EXCL}}} \left(1 - \frac{\lambda N_L P_{\text{EXCL}} T_{\text{H}}}{L}\right)^{N_L (1 - P_{\text{EXCL}})}.$$

An approximation similar to that in Eq. (2.7) is

$$P_{\text{CONT}} \simeq \frac{\lambda N_L^2 T_{\text{H}} P_{\text{EXCL}} (2 - P_{\text{EXCL}})}{L}. \tag{2.16}$$

Under the OCC environment, a similar development leads to

$$P_{\text{A}} \simeq \frac{\lambda N_L^2 T_{\text{H}} P_{\text{EXCL}}}{L}. \tag{2.17}$$

2.2. Additive Approximation and Conservation Property of Conflict. The approximate expressions for the transaction abort/invalidation probability when using the generic OCC scheme given in Eq. (2.11) and for the transaction contention probability when using the generic locking CC scheme given in Eq. (2.7) are the same for the case of a single transaction type with exclusive access

mode. (These approximations contain only the dominant term, which is $O(1/L)$ in the approximations in Eqs. (2.10) and (2.6), respectively. However, we believe that the $O(1/L)$ expressions provide insight into the effects of data conflicts for the various CC schemes we consider, and thus we discuss them in this section. When we present numerical results in Section 3.3, we use the approximations in Eqs. (2.6) and (2.10) that we have found to be more accurate at high levels of data contention than the $O(1/L)$ approximations.) This is true even if the database consists of multiple sets of granules with different access probabilities as shown in Eqs.(2.14) and (2.15). Note, however, that the mean granule holding times will in general not be equal for the two schemes, as discussed later in this section, so that the conflict probabilities will differ.

Considering the case of multiple transaction types, we find that for a particular transaction type, the $O(1/L)$ approximations for the transaction abort/invalidation probability using OCC and transaction contention probability using locking are still quite similar, as shown in Eqs. (2.12) and (2.13). They are both expressed as the sums of conflict terms from different sources. We refer to this as the *additive approximation.* The difference between Eqs. (2.12) and (2.13) is that the invalidation probability of a type j transaction due to type i transactions is proportional to T_{H_j} whereas the contention probability of a type j transaction due to type i transactions is proportional to T_{H_i}. Even so, since the overall conflict probability is the sum of conflict probabilities over all transaction types weighted by the relative throughputs, λ_j/λ, of the corresponding transaction types, we get the same approximate expression for overall conflict probability for both CC schemes. The rationale is as follows: Each term in the conflict expression represents a type of conflict. Depending upon the CC scheme, optimistic vs. locking, different actions can occur as to which transaction gets penalized and whether to wait or abort, but the different CC schemes do not eliminate any type of conflict. We refer to this as the *conservation property.* In analyzing CC schemes, including hybrid schemes that use both locking and optimistic methods, we have to decide which types of conflict contribute to abort and which contribute to contention wait, and add up the components to approximate the desired abort or wait probability. Further discussion appears in later sections when we explore different CC schemes.

Even though the approximate expressions for contention and abort probabilities have the similarities described above, the CC schemes do affect the mean granule hold time, T_H, in very profound ways. In the optimistic case, the abort probability affects the number of reruns and hence the CPU utilization which affects T_H. In the dynamic locking case, the contention probability increases the transaction response time, and consequently, the mean granule holding time, T_H. If the CPU resource is unlimited, the optimistic approach provides lower conflict since T_H is close to the processing time. If the CPU resource is very limited, static locking tends to provide lower conflict. (In static locking, a transaction tries to acquire all locks at once at the beginning. If it cannot get all of them, it does not hold on to any locks, but tries later or waits until they all become available.) In this case, the wait time to get the locks does not enlarge the granule holding time. In [42], a CC scheme using a combination of locking and OCC is proposed, and is shown to improve the performance over both OCC and locking schemes by striking a balance between the effect of transaction aborts and lock waits.

Finally, a word of caution. If we allow for different access modes like shared and exclusive, the conservation property may no longer hold. This is clear by comparing Eqs. (2.16) and (2.17). This is due to the fact that the concept of access modes is a further optimization to reduce conflicts between transactions and different CC schemes like locking and OCC, depending upon the specific implementation, can have different capabilities to exploit it. If we make the following changes to locking, the two schemes will again follow the conservation property. An intent-exclusive mode is now introduced in locking. An update access will first request an intent-exclusive lock and upgrade the lock mode to exclusive at commit time. Intent-exclusive mode is assumed to be compatible with shared mode, but incompatible with exclusive mode and intent-exclusive mode. Furthermore, shared mode is also assumed to be compatible with intent-exclusive mode. The advantage of introducing an intent-exclusive mode is to eliminate some unnecessary conflicts between shared and exclusive accesses under the original locking implementation. Now a shared lock does not wait (as the only incompatible mode, the exclusive mode, has an infinitesimal holding time under the assumption of zero commit time), and an exclusive mode lock may have to wait for shared mode locks to be released, which is the only type of shared-exclusive conflict. Similar to the derivation at the end of the previous section the lock contention probability with shared and intent-exclusive modes can be derived and verified to equal to that of OCC. Another case where the conservation property is violated is when transactions that may not be successfully committed abort conflicting transactions, or when transactions wait for subsequently aborted transactions, as in wound wait [29], locking with no-waiting [36], various running priority schemes in [16], and wait depth limited concurrency control in [18]. These schemes produce unnecessary aborts or waits. On the other hand, these schemes may result in smaller values for T_H compared to locking.

3. *Application to Centralized Schemes*

We apply the methodology just presented to approximate the mean response time for optimistic and locking CC schemes for a centralized database system. Unless stated otherwise, for simplicity, we assume that there is only one transaction type, the granule access distribution is uniform, and all accesses are exclusive.

3.1. Optimistic Protocols.

In this subsection, we demonstrate how to apply the proposed methodology to analyze different OCC schemes. Specifically we analyze the *pure OCC* scheme where a transaction is aborted only at certification time, and the *broadcast OCC* scheme where a transaction is aborted as soon as any granule it has accessed is made obsolete by a committing transaction. Previous analyses ignored the fact that, due to buffering in main memory, rerun transactions may not need to reaccess from the disk all the granules brought in during previous runs, and therefore came out in favor of broadcast OCC. Our analysis specifically captures this effect and is able to show that pure OCC can, in fact, outperform broadcast OCC. (The property that reruns of a transaction access the same granules as during the first run even though the runs are separated by those of conflicting transactions was called access invariance in [17]. Analytic modeling of other CC schemes that exploit access invariance can be found in [43].) Note that both pure OCC and

broadcast OCC schemes attempt to serialize transactions using certification times as the timestamps [2]. The methodology can be extended to handle other OCC certification schemes using dynamically derived timestamps or interval of timestamps [3, 47].

3.1.1. PURE OPTIMISTIC. We first show how the abort probability can be approximated, then present the hardware system resource model and show how the two models can be coupled together. The case of fixed-length transactions is considered first, and then the methodology is generalized to handle variable-length transactions. Validation is provided in Section 3.3.

3.1.1.1. *Abort Probability Analysis for Fixed-Length Transactions.* In Section 2.1, when we analyzed the pure OCC scheme, we assumed that commit processing was instantaneous. We now approximate the abort probability of pure OCC in a more realistic setting. We assume that each transaction at the beginning of each state of its execution phase informs the CC manager of its request to access a new granule. The CC manager keeps a list of transactions accessing each granule, and also maintains locks for granules held by transactions in commit. If a transaction requests access to a granule for which an incompatible lock is held by a transaction in commit, it is marked for abort. At commit time, the CC manager checks if a transaction has been marked for abort. If so, the CC manager removes the transaction from the list of granules accessed, and the transaction restarts after a backoff interval with mean duration T_{Backoff}. Otherwise, the transaction enters commit processing, and the CC manager grants locks for the granules accessed by the transaction and marks for abort any transactions that have conflicting access. The transaction then writes commit records to the log and propagates the updates to the disk, modeled as taking a mean time of T_{Commit}, following which locks are released. Let N_L denote the (fixed) number of granules accessed by a transaction. The initial setup phase of the transaction consists of execution of a mean of P_{INPL} instructions and a mean of I_{INPL} I/Os. In the first run of a transaction, the mean time in each state i, $1 \le i \le N_L$, is assumed to be the same and is denoted by $\hat{R}$. $\hat{R}$ corresponds to execution of a mean of $\hat{P}$ instructions, and a mean of $(1 - p_{h1})$ I/Os, where p_{h1} denotes the (buffer hit) probability that the accessed granule is found in a main memory buffer. In a rerun of a transaction due to an abort, the mean time in each state i, $1 \le i \le N_L$, is assumed to be the same but different from that in the first run. It is denoted by R', and corresponds to a mean of P' instructions, and $(1 - p_{h2})$ I/Os, where p_{h2} denotes the buffer hit probability during a rerun.

Assuming instantaneous commit processing, Eq. (2.10) approximates the invalidation probability on the first run P_A as $1 - \exp(-\lambda N_L^2 T_H/L)$, where T_H is the mean granule holding time and is equal to $(N_L + 1)\hat{R}/2$. This corresponds to the probability that a transaction is marked for abort by a transaction that enters commit processing after the invalidated transaction accessed a conflicting data granule. Recall that transactions are also aborted when the new granule accessed at any stage is already locked by a transaction in commit; the probability that this occurs is approximated by Eq. (2.6) with T_{Commit} replacing T_H. This latter case corresponds to the probability that the invalidated transaction accesses a data granule after the conflicting transaction enters commit processing. Therefore, transactions (in the process of commit) that hold a lock on the new granule requested and transactions that initiate

commit during the ith state of the transaction being considered are different sets of transactions. We assume that invalidations due to these two cases are independent events since they arise from conflicts from different sources. Thus, the probability of transaction abort is approximated as,

$$P_A \simeq 1 - \exp\left(\frac{-\lambda N_L^2 T_H}{L}\right)\left(1 - \frac{\lambda N_L T_{Commit}}{L}\right)^{N_I}. \tag{3.1}$$

This can be further approximated as

$$P_A \simeq \frac{\lambda N_L^2 (T_H + T_{Commit})}{L}. \tag{3.2}$$

Although we can derive Eq. (3.2) from Eq. (3.1), what is more interesting is that *we can directly write down the expression based on the additive approximation discussed in Section* 2.2. In doing this, it is important to identify the sources of abort. In this case there are two: one is from invalidation, and the other is from lock contention. Taking advantage of the additive approximation, we can estimate each term separately based on Eqs. (2.7) and (2.11) and sum them.

In a similar manner, the probability of abort P'_A for a rerun transaction is approximated by replacing T_H in Eqs. (3.1) and (3.2) by the average granule holding time $T'_H (= (N_L + 1)R'/2)$ in a rerun.

3.1.1.2. *Hardware Resource Model.* We now describe the hardware resource model and show how it can be coupled with the data contention model through an iteration. We assume that the system consists of K tightly coupled processors and a database that is spread over multiple disks. Assuming Poisson arrivals of transactions, we model the processors as an M/M/K queue with FCFS discipline and the disks as an infinite server. Other open queuing network models could be used; we chose this one for simplicity. The processor utilization is given by

$$\rho = \frac{\lambda\left\{P_{INPL} + N_L \hat{P} + P_A/(1 - P'_A) N_L P'\right\}}{K \times \text{MIPS}}, \tag{3.3}$$

where MIPS is the processor speed. In the above, the probabilities of abort are estimated in terms of the mean times in each state $\hat{R}$ and R', and the utilization ρ is expressed in terms of the abort probabilities. Now, based on M/M/K results [26], $\hat{R}$ and R' are given as follows: Define,

$$\alpha = \frac{(K_\rho)^K}{K!(1 - \rho)}; \qquad \beta = \sum_{J=0}^{K-1} \frac{(K\rho)^J}{j!}. \tag{3.4}$$

Then, the mean CPU response time, R_x^C, for a state with execution of PL_x instructions is given by,

$$R_x^C = \gamma \times \frac{PL_x}{\text{MIPS}}, \tag{3.5}$$

where

$$\gamma = 1 + \frac{\alpha}{K(\alpha + \beta)(1 - \rho)}. \tag{3.6}$$

Then, the mean times in different states are given as,

$$\begin{aligned} R_{\text{INPL}} &= \gamma \times \frac{P_{\text{INPL}}}{\text{MIPS}} + I_{\text{INPL}} \times \text{IOTIME}, \\ \hat{R} &= \gamma \times \frac{\hat{P}}{\text{MIPS}} + \hat{I} \times \text{IOTIME}, \\ R' &= \gamma \times \frac{P'}{\text{MIPS}} + I' \times \text{IOTIME}, \end{aligned} \tag{3.7}$$

where $\hat{I} = 1 - p_{h1}$ (respectively, $I' = 1 - p_{h2}$) is the average number of I/Os per state in the first run (respectively, any subsequent run). Making the same four assumptions that were made prior to Eq. (2.3), it follows that the overall mean transaction response time is given by

$$R = R_{\text{INPL}} + N_L \hat{R} + T_{\text{Commit}} + \frac{P_A}{(1 - P'_A)} \times \{T_{\text{Backoff}} + N_L R'\}. \tag{3.8}$$

From Section 3.1.1.1, we know P_A and P'_A depend upon $\hat{R}$ and R', respectively, whereas $\hat{R}$ and R' depend upon the CPU utilization ρ which in turn depends upon P_A and P'_A. There clearly is an interdependency. We can first pick some arbitrarily small abort probabilities, P_A and P'_A, and calculate the mean response times, $\hat{R}$ and R'. The mean response times are then used to calculate a new set of abort probabilities. The process continues and typically converges in a few iterations.

3.1.1.3. *Variable-Length Transactions.* For variable-length transactions, the mean granule holding time T_{H} for first-run transactions can be expressed as follows. For transaction j, let n_j be the number of granules accessed, and r_{kj} be the time in state k. Then, since $t_{ij} = \sum_{k=i}^{n_j} r_{kj}$, it follows from Eq. (2.9) that

$$\begin{aligned} T_H &= \frac{1}{N_L} \lim_{J \to \infty} \frac{1}{J} \sum_{j=1}^{J} \sum_{i=1}^{n_j} \sum_{k=i}^{n_j} r_{kj} = \frac{1}{N_L} \lim_{J \to \infty} \frac{1}{J} \sum_{j=1}^{J} \sum_{k=1}^{n_j} (k r_{kj}) \\ &= \frac{1}{N_L} \lim_{J \to \infty} \frac{1}{J} \sum_{j=1}^{J} E\left[\sum_{k=1}^{n_j} k r_{kj} \right]. \end{aligned}$$

Assuming that n_j is independent of $r_{1j}, r_{2j}, \ldots$, and that $r_{1j}, r_{2j}, \ldots$ are independent and identically distributed as a random variable r_j, then

$$E\left[\sum_{k=1}^{n_j} k r_{kj} \right] = \sum_{n=1}^{\infty} P\{n_j = n\} \sum_{k=1}^{n} k E[r_j] = E\left[\frac{n_j(n_j + 1)}{2} \right] E[r_j] = \frac{N_L^{(2)} + N_L}{2} \hat{R},$$

where $N_L^{(2)}$ is the second moment of n_L, and $\hat{R}$ is the mean time in each execution state on the first run. Thus,

$$T_{\text{H}} = \frac{N_L^{(2)} + N_L}{2N_L} \hat{R}. \tag{3.9}$$

In analyzing the performance with variable-length transactions, the probability of abort on the first run can be approximated using Eq. (3.1) or (3.2) with

T_{H} expressed above in terms of the first and second moments of n_L. The problem with this approach is that since the distribution of the lengths of rerun transactions does not in general equal that of first-run transactions (unless transaction lengths are constant), it does not yield the rerun transaction abort probabilities. In particular, long transactions are more likely to get aborted, thus skewing the distribution of rerun transactions to longer length than first-run transactions. In turn, this leads to higher CPU utilizations and consequently affects the first-run abort probabilities as well.

We therefore handle variable-length transactions by considering different transaction classes where the transactions in a given class have fixed length. For simplicity, we assume that transactions in different classes differ only in their lengths, that is, number of granules accessed. Suppose that transactions belong to class i with probability P_i and access N_{L_i} granules, that is, $P_i = \Pr\{n_L = N_i\}$, and the P_i are assumed to be known. Then, analogous to Eq. (3.1), the probability of abort P_{A_i} of first-run class i transactions can be approximated as

$$P_{A_i} \simeq 1 - \exp\left(\frac{-\lambda N_L T_{H_i} N_{L_i}}{L}\right)\left(1 - \frac{\lambda N_L T_{\mathrm{Commit}}}{L}\right)^{N_{L_i}}, \tag{3.10}$$

where $N_L = \Sigma_i P_i N_{L_i}$ is the average number of granules accessed per transaction. In this equation, the probability of abort of class i transactions depends on the average number of granules accessed by committing transactions and on the number of granules accessed by class i transactions. Since we have assumed the system is stable, the rate at which each class of transactions commits is equal to the arrival rate to that class. Thus, the class composition of committing transactions is equal to that of arriving transactions and the average number of granules accessed by committing transactions equals N_L. Similarly, P'_{A_i} is approximated by replacing T_{H_i} by T'_{H_i}, in Eq. (3.10). Then, analogous to Eq. (3.3),

$$\rho = \sum_i \frac{\lambda P_i\left\{P_{\mathrm{INPL}} + N_{L_i}\hat{P} + \left(P_{A_i}/(1 - P'_{\mathrm{A}_i})\right)N_{L_i}P'\right\}}{K \times \mathrm{MIPS}}.$$

Then $\hat{R}$ and R' are approximated using Eq. (3.7), and for class i transactions, Eq. (3.8) becomes

$$R_i = R_{\mathrm{INPL}} + N_{L_i}\hat{R} + T_{\mathrm{Commit}} + \frac{P_{\mathrm{A}_i}}{1 - P'_{\mathrm{A}_i}}\{T_{\mathrm{Backoff}} + N_{L_i}R'\}. \tag{3.8}'$$

3.1.2. BROADCAST OCC. We first present the analysis for a fixed number of granules accessed per transaction and then generalize this to variable-length transactions.

3.1.2.1. *Fixed-Length Transactions.* Let N_L be the (fixed) number of granules accessed by a transaction. Under broadcast OCC, the CC manager is assumed to keep a list of transactions marked for abort, and abort the transaction the next time it notifies the CC manager of a new granule access. The transaction model for this case is thus similar to that for pure OCC except that a transaction may abort (return to state 1) at the time it is ready to transit

from state i to state $i + 1$, that is, at the time it notifies the CC manager on accessing a new granule. The *phase* of a transaction refers to the highest numbered state that the transaction has ever reached in the *previous* runs of the transaction. A transaction starts in phase 0, indicating that no data granules have been referenced. If in the first run it acquires j data granules and then aborts (due to a conflict detected by the CC manager when it is notified on the access of the $(j + 1)$th data granule) then it is in phase j when it restarts. In phase j, the I/Os corresponding to states 1 through j have already been done during some previous phase, and thus it is reasonable to assume that these granules are more likely to be in the buffer than they would otherwise be. Thus, we assume for phase j that the buffer hit probability for states $1,\ldots,j$ differs from that for states $j + 1,\ldots,N_L$. Making the same independence assumption that preceded Eq. (3.1), the conditional probability of abort $A(j,i)$ at the end of the ith state of the jth phase given that the transaction was not aborted previously during phase j is approximated as,

$$A(j,i) \simeq \begin{cases} 1 - \exp(-iN_L \lambda R'/L)\left(1 - \dfrac{N_L \lambda T_{\text{Commit}}}{L}\right) & \text{for } i \le j \\ 1 - \exp\left(-iN_L \lambda \hat{R}/L\right)\left(1 - \dfrac{N_L \lambda T_{\text{Commit}}}{L}\right) & \text{for } i > j. \end{cases} \tag{3.11}$$

Here the abort probability in each state is computed separately, and the mean time in the state is either $\hat{R}$ or R' depending on the buffer hit probability. Recall that in the pure OCC scheme, we assumed the buffer hit probability during the first run differed from that during all subsequent runs. Let $C(j,i)$, for $1 \le i \le N_L$, be the unconditional probability of getting invalidated at state i for a transaction at phase j, and let $C(j, N_L + 1)$ be the probability of being successfully committed. Then, assuming independence of the invalidation events for different states,

$$C(j,i) = \left\{\prod_{k=1}^{i-1}(1 - A(j,k))\right\}A(j,i), \qquad 1 \le i \le N_L,$$

$$C(j, N_L + 1) = \prod_{k=1}^{N_L}(1 - A(j,k)). \tag{3.12}$$

After simplification, we can get the approximations for $1 \le i \le j \le N_L$,

$$C(j,i) \simeq \left(1 - \frac{\lambda N_L \Sigma_{k=1}^{i-1} kR' + (i-1)T_{\text{commit}}}{L}\right)\frac{\lambda N_L(iR' + T_{\text{commit}})}{L}$$

and similar approximations for $1 \le j < i \le N_L$ and for $C(j, N_L + 1)$. Since $\lambda N_L(kR' + T_{\text{commit}})/L$ is the $O(1/L)$ approximation to the probability of invalidation at stage k, this approximation for $C(j,i)$ can be written down directly based on the additive approximation discussed in Section 2.2.

The remaining mean response time at the start of phase j can be approximated using a set of difference equations as,

$$B(j) = \left\{ \sum_{i=1}^{J} C(j,i)\{B(j) + T_{\text{Backoff}} + iR'\} \right\} + \left\{ \sum_{i=j+1}^{N_L} C(j,i)\{B(i) + T_{\text{Backoff}} + jR' + (i-j)\hat{R}\} \right\} + C(j, N_L + 1)\{T_{\text{Commit}} + jR' + (N_L - j)\hat{R}\}. \quad (3.13)$$

The first summation term in this equation corresponds to a transaction aborting after advancing to a state with number less than or equal to the current phase of the transaction. The ith term in the summation computes the probability of no abort up to entering state i, and an abort upon leaving state i, and multiplies this probability with the sum of the remaining mean response time after the abort, the mean backoff time, and the mean execution time through the end of state i. Since the case is for an abort before the state number exceeds the phase, the transaction stays in the same phase, and the remaining mean response time is $B(j)$. The second summation term in Eq. (3.13) corresponds to the case when a transaction aborts after advancing to a state number greater than its current phase. The ith term of this second summation is similar to the ith term of the first summation, except that the first j states and the subsequent $(i-j)$ states have different buffer hit probabilities, and the remaining mean response time after the abort is $B(i)$ since the transaction would advance to phase i (greater than j). The final term in the equation is for the case when a transaction completes and commits. Now, $B(0)$ approximates the mean transaction response time, excluding the mean initial set-up time R_{INPL}.

The processor utilization can be approximated using a similar equation to (3.13) as follows: Let $D(j)$ be the mean number of remaining instructions executed by the transaction at the start of phase j. Then,

$$D(j) = \left\{ \sum_{i=1}^{J} C(j,i)\{D(j) + iP'\} \right\} + \left\{ \sum_{i=j+1}^{N_L} C(j,i)\{D(i) + jP' + (i-j)\hat{P}\} \right\} + C(j, N_L + 1)\{jP' + (N_L - j)\hat{P}\}. \quad (3.14)$$

This equation is similar to (3.13) with $B(j)$ replaced by $D(j)$, $\hat{R}$ by $\hat{P}$ and R' by P'. Thus, $D(0)$ approximates the mean number of instructions executed by a transaction, excluding P_{INPL}. The processor utilization is approximated as,

$$\rho = \frac{\lambda\{D(0) + P_{\text{INPL}}\}}{K \times \text{MIPS}}. \quad (3.15)$$

Now, the constants α, β, γ are obtained using Eqs. (3.4) and (3.6), and R_{INPL}, $\hat{R}$ and R', by Eq. (3.7). The mean response time R is given by $\{R_{\text{INPL}} + B(0)\}$.

Again, an iteration is needed to handle the dependencies among the above equations.

3.1.2.2. *Variable-Length Transactions.* The extension of the analysis to variable-length transactions is analogous to that in Section 3.1.1.3, and assumes that with probability P_m transactions belong to class m and have fixed-length N_{L_m}. Then, all of the development in Section 3.1.2.1 goes through (except for Eq. (3.15)) with the equations for $A(j,i)$, $C(j,i)$, $B(j)$, and $D(j)$ replaced by corresponding equations for $A_m(j,i)$, $C_m(j,i)$, $B_m(j)$, and $D_m(j)$ where m is the transaction class and for the equations with subscript m, $1 \le i, j \le N_{L_m}$. Note from Eq. (3.11) and the above that the abort probability for class m transactions $A_m(j,i)$, at the end of the ith state of the jth phase, depends on the granule access rate by committing transactions, λN_L, and not merely on $\lambda P_m N_{L_m}$. Hence, all the N_L in Eq. (3.11) remain unchanged for $A_m(j,i)$. However, in Eqs. (3.12), (3.13), and (3.14), all N_L should be replaced by N_{L_m} for approximating $C_m(j,i)$, $B_m(j)$, and $D_m(j)$. Then analogous to Eq. (3.15) the CPU utilization is approximated as,

$$\rho = \sum_m \frac{\lambda P_m \{D_m(0) + P_{\text{INPL}}\}}{K \times \text{MIPS}}. \tag{3.15'}$$

A variation of the broadcast OCC scheme considered in [43] is as follows. The transaction uses pure OCC for the first run of the transaction, and uses the early abort of broadcast OCC for any subsequent reruns. This is referred to as OCC with broadcast during rerun. It can be similarly analyzed following the approach in Section 3.1.2 with all rerun transactions now in phase N_L.

3.2. Standard Locking. As in the previous sections, we first consider fixed-length transactions and then generalize this to variable-length transactions.

3.2.1. Fixed-Length Transactions. We now show how the methodology can be applied to analyze the standard two-phase locking (2PL) scheme. Let N_L be the (fixed) number of granules accessed by a transaction. In our transaction model, transactions make lock requests at the beginning of states $1, \ldots, N_L$. If the granule has already been locked in an incompatible mode, the transaction is enqueued at the CC manager and waits till the lock becomes available. As before, for locking we neglect the probability of deadlock. For locking each state i, $1 \le i \le N_L$, is divided into two substates $\tilde{i}$ and $\hat{i}$. In substate $\tilde{i}$ the transaction holds $i-1$ locks and is waiting for its ith lock request to be satisfied. In substate $\hat{i}$, it holds i locks and is executing. Let a denote the mean time in substate $\hat{i}$ that is assumed to be independent of i and is obtained from the hardware resource model as discussed later in this section. The probability that substate $\tilde{i}$ is entered upon leaving substate $\hat{i}-1$ is the lock contention probability for a lock request, P_W, which is given with Eq. (2.4) and assumed to be independent of i. The time spent in substate $\tilde{i}$, given it is entered, is the lock waiting time whose mean we denote by R_W which is also assumed to be independent of i. The unconditional mean time in substate $\tilde{i}$ is therefore $b = P_W R_W$. R_W can be approximated as follows: The mean number of transactions in substate $\tilde{i}$ (respectively, $\hat{i}$) is λb (respectively, λa). Since a transaction in substate $\tilde{i}$ (respectively, $\hat{i}$) holds $i-1$ locks and is waiting for its

ith lock (respectively holds i locks) and the granule access distribution is uniform, the probability that a lock request contends with a transaction in substate $\tilde{i}$ (respectively, $\hat{i}$) is $(i-1)\lambda b/L$ (respectively, $i\lambda a/L$), $1 \le i \le N_L$. Similarly, the probability that a lock request contends with a transaction in state $N_L + 1$ is $\lambda N_L c/L$ where $c = T_{\text{Commit}}$. Let

$$G = \left\{ \sum_{i=1}^{N_L} (ia + (i-1)b) \right\} + N_L c. \tag{3.16}$$

(Note that since i locks (respectively, $i-1$ locks) are held in substate $\hat{i}$ (respectively, substate $\tilde{i}$) with a mean sojourn time of a (respectively, b), G is the sum of the (mean) lock holding time for each granule over all N_L granules, and G/N_L is the mean lock holding time averaged over all N_L granules.) Then,[5]

$$R_W \simeq \sum_{i=1}^{N_L} \left(\frac{(i-1)b}{G} \left\{ \frac{R_W}{f_1} + a + s_i \right\} + \frac{ia}{G} \left\{ \frac{a}{f_2} + s_i \right\} \right) + \frac{N_L c}{G} \left\{ \frac{c}{f_3} \right\}. \tag{3.17}$$

In the above expression, $(i-1)b/G$ (respectively, ia/G) is the conditional probability that a lock request contends with a transaction in substate $\tilde{i}$ (respectively, $\hat{i}$) given that lock contention occurs, $1 \le i \le N_L$, and $N_L c/G$ is the similar expression for state $N_L + 1$. The quantity s_i is the mean time from leaving state $\hat{i}$ until the end of commit and is given by

$$s_i = (N_L - i)(a + b) + c. \tag{3.18}$$

The quantity R_W/f_1 (respectively, a/f_2) is the mean remaining time in substate $\tilde{i}$ (respectively, $\hat{i}$) given that the transaction contended with was in that state, $1 \le i \le N_L$ and similarly for c/f_3. Note that in obtaining Eq. (3.17), we have assumed that when a transaction T_1 encounters contention for a lock held by another transaction T_2, then no other transaction T_3 can be waiting for this same lock, that is, the queue length for any lock never exceeds two. Thus, when T_2 commits and releases the lock T_1 acquires it without further waiting. Note, however, that T_2 can be waiting for a lock held by yet another transaction T_4, and T_4 can also be waiting, etc. Thus, wait chains can build up due to waits for different locks, and this is captured by the analysis. Since transactions typically hold several locks, wait chains of this type (rather than for the same granule) predominate. The factors f_1, f_2, and f_3 depend on the distributions of the times in substates $\tilde{i}$, $\hat{i}$, and $N_L + 1$, respectively. Substate $\hat{i}$ corresponds to the CPU execution time and the I/O time if the granule accessed in the state is not found in memory (recall that p_{h1} denotes the probability that a granule accessed in any state is found in a main memory buffer). Assuming that p_{h1} is not very close to one and that I/O time is constant and much larger than the processing time (this is true for typical parameters such as those used in Section 3.3), then it is easy to show that $f_2 \simeq 2(1 - p_{h1})$. In our studies, we assume that $f_3 = 2$ corresponding to a constant time in the the commit state and that $f_1 = 1$ corresponding to exponential lock-waiting times. Comparison

[5] An independently derived approximation that appears in [40] for multiple transaction classes, with fixed-length transactions in each class, reduces to a similar expression if there is only one class. However, for multiple classes his approximation differs from the one we give in the next section.

with simulations [41, 42] indicated that $f_1 = 1$ yields good results. At low contention levels, the estimates from $f_1 = 1$ are a little pessimistic. Equation (3.17) can be simplified to yield,

$$R_W \simeq \frac{(a+b)^2(N_L+1)(N_L-1)}{6} \Bigg/ \left\{ a\left(\frac{N_L+1}{2}\right) + c + b\left(\frac{N_L-1}{2}\right)\left(1-\frac{1}{f_1}\right)\right\}$$
$$+ \frac{(N_L+1)\left((a^2/f_2)+ab+ac+bc\right)}{2} \Bigg/ \left\{ a\left(\frac{N_L+1}{2}\right) + c + b\left(\frac{N_L-1}{2}\right)\left(1-\frac{1}{f_1}\right)\right\}$$
$$+ \left\{\frac{c^2}{f_3} - ab - bc\right\} \Bigg/ \left\{ a\left(\frac{N_L+1}{2}\right) + c + b\left(\frac{N_L-1}{2}\right)\left(1-\frac{1}{f_1}\right)\right\}. \tag{3.19}$$

For large N_L, small c, and small b compared to a, the right side of Eq. (3.19) is approximately equal to $((a+b)(N_L-1)+c)/3$. This is the approximation used in [46]. Furthermore, it follows from using Eq. (2.4) and computing the mean lock holding time T_H from Eq. (3.16) that

$$P_W = \frac{\lambda G}{L}. \tag{3.20}$$

From Eq. (2.2), the mean response time is

$$R = R_{INPL} + R_E + N_L P_W R_W + T_{Commit}, \tag{3.21}$$

where $R_E = N_L \times a$ is the mean duration of the execution phase, which can be approximated using a hardware resource model similar to that in Section 3.1.1.2. Since both R_W and P_W in Eqs. (3.19) and (3.20), respectively, depend upon $b = P_W R_W$, we need to solve for them through an iteration. Starting with $b = 0$, we can get initial values for R_W and P_W, and thus a new b value. The process continues and typically converges in a few iterations.

Equation (3.19) can also be used to estimate the probability of lock contention at which the mean response time blows up. For large N_L, $(a+b)$ much larger than c (which occurs as the waiting time increases), and $f_1 = 1$ (as discussed above), Eq. (3.19) reduces to

$$R_W = \frac{b}{P_W} \simeq \frac{(a+b)^2(N_L-1)}{3a}. \tag{3.22}$$

Define $P_W^T = P_W(N_L - 1)$, which approximates the average number of contentions per transaction for large N_L. Equation (3.22) can be solved for b to yield,

$$b = \frac{a}{2}\left(\frac{3}{P_W^T} - 2 - \sqrt{\left(2-\left(3/P_W^T\right)\right)^2 - 4}\right). \tag{3.23}$$

(Note that the negative sign is required for the term under the square root, so that $b' \to 0$ as $P_W^T \to 0$.) From Eq. (3.23), the maximum value of the mean number of contentions per transaction P_W^T beyond which there is no solution for b is $P_{W\_MAX}^T = 0.75$.

3.2.2. VARIABLE-LENGTH TRANSACTIONS. The extension of the analysis to variable-length transactions, as in the analysis of OCC schemes, assumes that with probability P_m transactions belong to class m and access a fixed number of granules, N_{L_m}. Again, for simplicity, other transaction characteristics, that is, mean execution time in a state and mean commit time, are assumed not to differ between classes. Let P_{W_m} be the probability of contention with a class m transaction, and let R_{W_m} be the mean waiting time given that there was a contention with a class m transaction. These quantities are assumed to be independent of the requesting transaction. Then, the overall contention probability P_W and overall mean waiting time R_W are given by,

$$P_W = \sum_m P_{W_m}$$

and

$$R_W = \frac{\Sigma_m P_{W_m} R_{W_m}}{P_W}.$$

Let $b = \Sigma_m P_{W_m} R_{W_m}$ and

$$G_m = \left\{ \sum_{i=1}^{N_{L_m}} (ia + (i-1)b) \right\} + N_{L_m} c.$$

Let s_i^m be the mean time from leaving state $\hat{i}$ until the end of commit for a class m transaction. Then,

$$s_i^m = (N_{L_m} - i)(a + b) + c$$

and

$$R_{W_m} \simeq \sum_{i=1}^{N_{L_m}} \left(\frac{(i-1)b}{G_m} \left\{ \frac{R_W}{f_1} + a + s_i^m \right\} + \frac{ia}{G_m} \left\{ \frac{a}{f_2} + s_i^m \right\} \right) + \frac{N_{L_m} c}{G_m} \left\{ \frac{c}{f_3} \right\}. \tag{3.24}$$

Equation (3.23) can be simplified to yield,

$$\begin{aligned} R_{W_m} \simeq {} & \frac{(a+b)^2 (N_{L_m}+1)(N_{L_m}-1)}{6} \Bigg/ \left\{ a\left(\frac{N_{L_m}+1}{2}\right) + b\left(\frac{N_{L_m}-1}{2}\right) + c \right\} \\ & + \frac{(N_{L_m}+1)\left((a^2/f_2) + (bR_W/f_1) + ab + ac + bc\right)}{2} \Bigg/ \\ & \left\{ a\left(\frac{N_{L_m}+1}{2}\right) + b\left(\frac{N_{L_m}-1}{2}\right) + c \right\} \\ & + \left(\frac{c^2}{f_3} - \frac{bR_W}{f_1} - ab - ac \right) \Bigg/ \left\{ a\left(\frac{N_{L_m}+1}{2}\right) + b\left(\frac{N_{L_m}-1}{2}\right) + c \right\}. \end{aligned}$$

Thus, the mean response time of a class m transaction is given by

$$R_m = R_{\text{INPL}} + R_{\text{E}_m} + N_{L_m}\left(\sum_k P_{\text{W}_k} R_{\text{W}_k}\right) + T_{\text{Commit}}, \tag{3.25}$$

where $R_{\text{E}_m} = N_{L_m} \times a$.

3.3. VALIDATION AND PERFORMANCE COMPARISONS. The focus of this section is to validate the analysis methodology by comparison with estimates from a simulation model. We briefly discuss some comparative performance of the CC schemes. Further comparisons can be found in the references. We first consider the case of fixed-length transactions.

A discrete-event simulation was developed to validate the analysis. The simulation model is described in the Appendix. Included is a discussion of the simulation run lengths and the confidence intervals that were obtained and which were narrow in all cases. The following parameter values are used for illustrative purposes: the initial instructions executed (P_{INPL}) of 150K, initial I/Os (I_{INPL}) of 5, instructions executed ($\hat{P}$) in state i of 20K $\forall i$, average I/Os ($\hat{I}$) in state i of 0.73$\forall i$, that is, $P_{\text{h1}} = 0.37$, an average I/O time (IOTIME) of 0.035 seconds, fixed number of granules accessed per transaction (N_L) of 15, commit time (T_{commit}) of 0.025 seconds, and a backoff time $T_{\text{Backoff}} = T_{\text{commit}}$. Note that with these parameter values a transaction executes 450,000 instructions. The workload parameters are similar to the ones used in [46], which were derived from real workload measurements for a parts inventory and a material-planning database application. Fifteen granules accessed per transaction and 450,000 instructions executed per transaction are characteristic of middleweight transactions. We first examine the case for a database of 1000 granules and 5 tightly coupled processors of 10 MIPS each, $p_{h2} = 1$, that is, no repeated I/O during the rerun of an aborted transaction. Note that a relatively small database and hence relatively high data contention is chosen for this case so that we can better differentiate the robustness of the schemes and demonstrate the phenomena that cause the difference. Further, as described in Section 2, a hot set in the database can be modeled by uniform access to an effective database size, which we showed to be smaller than the original database. Therefore, the occurrence of hot-spots in the database leads to a comparatively small effective database size. Sensitivity to the database size parameter is considered later. With the hardware model we are using, the only other contention is for the CPUs. For a 10-MIPS processor, the CPU service time for a single complete run of a transaction is 0.045 seconds. For five such processors, if there were no data contention so that every transaction was run successfully to completion the first time, then the utilization of each processor would equal $(\lambda) \times (0.045/5) = 0.009\lambda$. Thus, the maximum transaction rate in the absence of data contention is approximately 111 transactions/second.

We first examine the accuracy of the approximations for probabilities of abort and contention. The probability of abort during the first run of a transaction is shown in Figures 1 and 2 for the cases of the pure OCC and the broadcast OCC schemes, respectively. The approximation in Eq. (3.1) is used for pure OCC. Although the $O(1/L)$ approximation in Eq. (3.2) is fairly accurate, it is less accurate than the approximation in Eq. (3.1) for high data contention. This is true in general for the $O(1/L)$ approximations, and we use

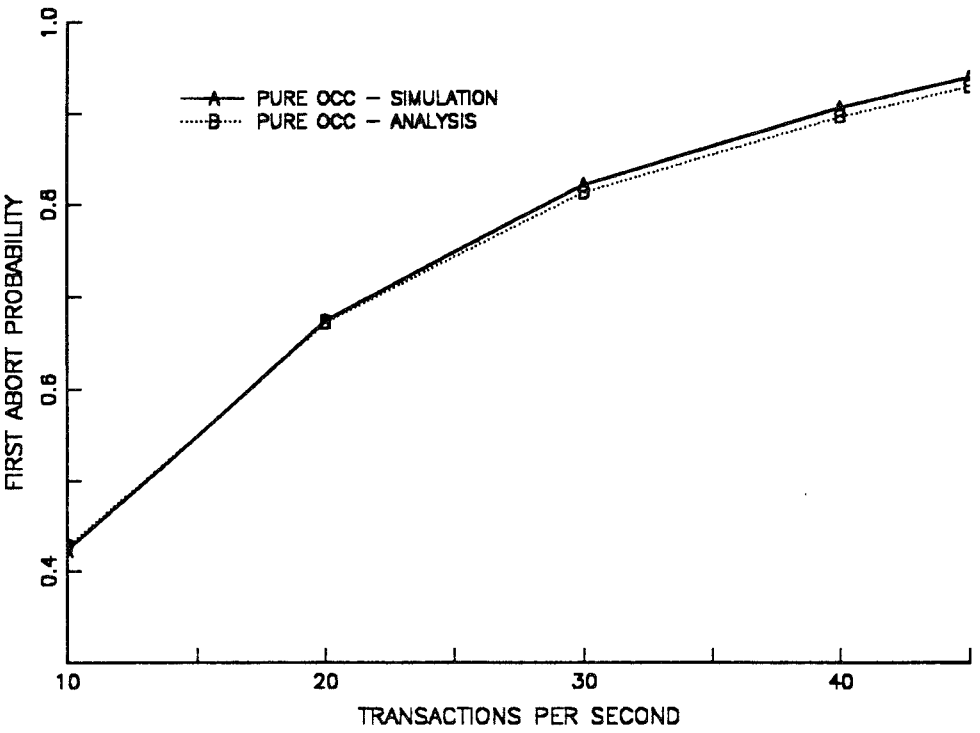

FIG. 1 First abort probability for Pure OCC.

the approximations that include the higher order terms in what follows. It follows from Eqs. (3.1) and (3.9), where $N_L^{(2)} = N_L^2$ for fixed-length transactions, that 1–2

$$P_A(\text{pure}) \simeq 1 - \exp\left(\frac{-\lambda N_L^2(N_L + 1)\hat{R}(\text{pure})}{2L}\right)\left(1 - \frac{\lambda N_L T_{\text{Commit}}}{L}\right)^{N_L}.$$

For broadcast OCC, the probability of abort during the the first run is equal to $1 - C(0, N_L + 1)$, where $C(0, N_L + 1)$ is obtained from Eqs. (3.11) and (3.12) so that

$$P_A(\text{broadcast}) \simeq 1 - \exp\left(\frac{-\lambda N_L^2(N_L + 1)\hat{R}(\text{broadcast})}{2L}\right)\left(1 - \frac{\lambda N_L T_{\text{Commit}}}{L}\right)^{N_L}.$$

Note that these two approximate expressions for P_A differ only in that the values of $\hat{R}$ can differ for the two schemes. $\hat{R}$ is affected by processor contention which in turn is affected by aborted transaction reruns. At very low transaction rates, where aborts are rare, the two values of $\hat{R}$ should be approximately equal for the two schemes, but this need not be the case at high transaction rates where aborts can be frequent. The figures show good agreement between the analytical approximations and simulation estimates even for very high abort probabilities. The curves for pure OCC and broadcast OCC are shown on separate figures since they largely overlap and are hard to distinguish if plotted on one chart. In fact, at low transaction rates, the first abort

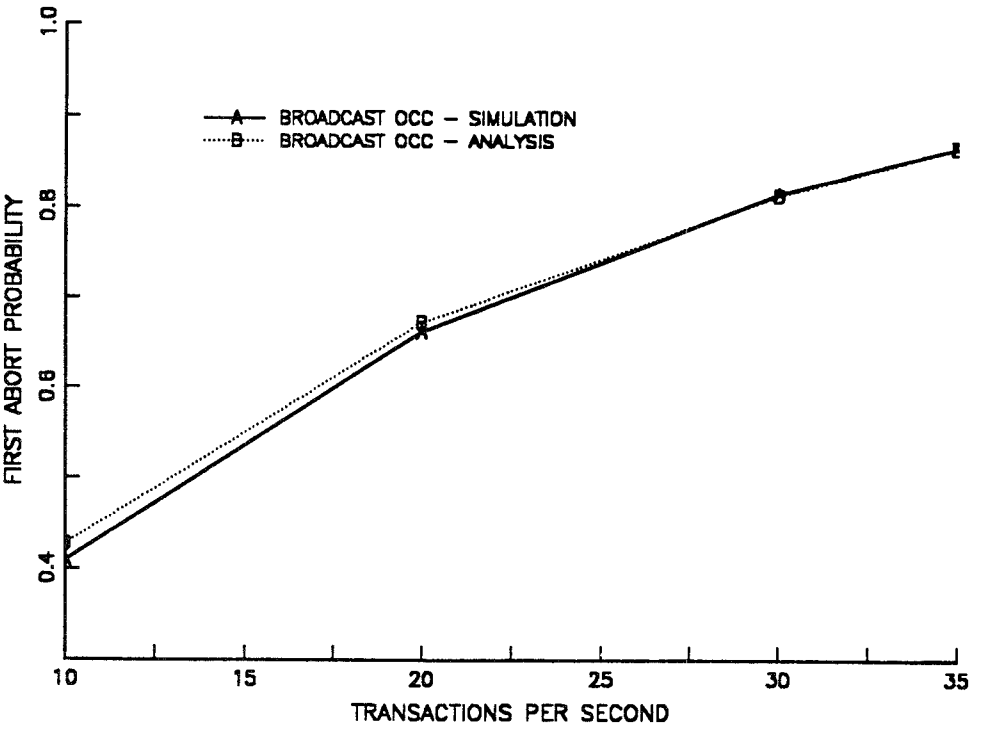

FIG. 2. First abort probability for Broadcast OCC.

probabilities of the two schemes are virtually identical as expected. At higher transaction rates, the first abort probability for broadcast OCC is slightly larger than that for pure OCC. (The reason for this will be discussed shortly when comparing the mean response times in Figure 4.) Figure 3 shows the probability of contention per lock request for the standard locking scheme using the approximations in Eqs.(2.4) and (3.19). The approximation compares fairly well with the simulation results. Note that in this figure the probability of contention per lock request is up to about 0.05, corresponding for the parameters chosen to a transaction contention probability of about 0.75, which is the limiting transaction contention probability derived in Section 3.2.1.

Figure 4 shows the mean transaction response time versus the transaction rate for the locking, pure OCC, broadcast OCC, and OCC with broadcast during rerun schemes. The approximation in Eq. (3.19) is used to estimate the mean lock waiting time for the locking case. Both analytical approximations and simulation estimates are provided. As described in Section 3, an iteration is used for each of the analytical approximations, and the iterative approximations were found to converge in a small number (usually less than ten) of iterations. There is good agreement between the simulation and the analysis. The analysis and simulation results for the cases of locking, pure OCC, and OCC with broadcast during rerun are very close even beyond the knee of the curves. For the broadcast OCC scheme, the analytical estimate is very close to the simulation estimate before the knee of the curves, and is a little optimistic beyond the knee, but accurately projects the location of the knee. The slightly larger discrepancy for the broadcast OCC estimates is due to the dependence

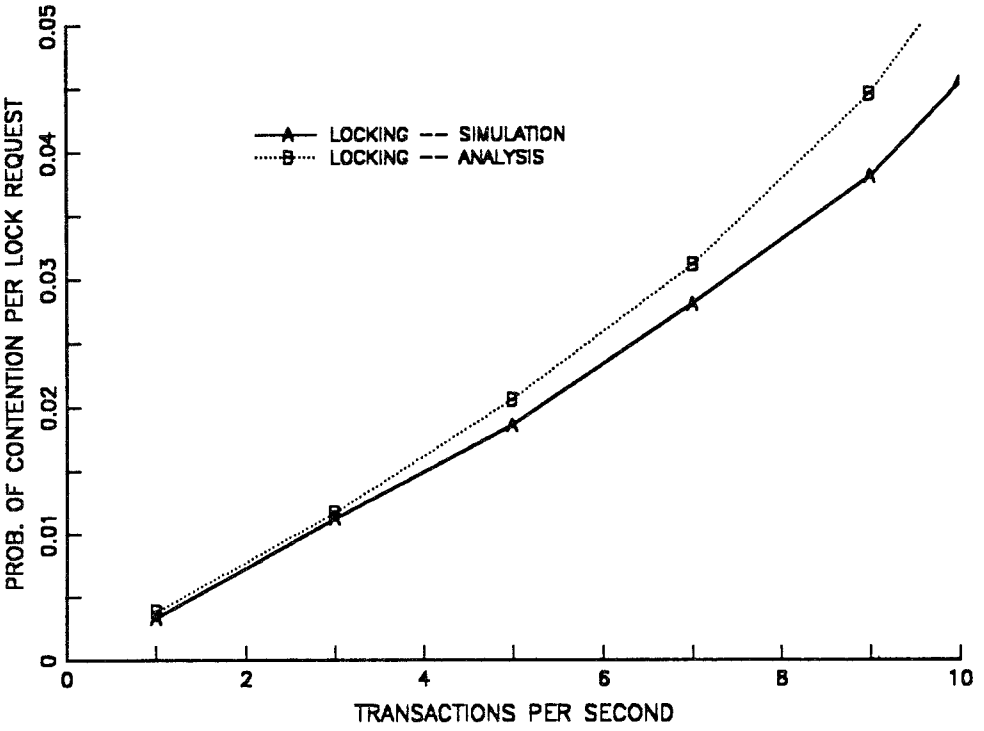

FIG. 3. Lock contention probability for 2-phase locking

between successive runs of an aborted transaction, where the number of I/Os in one run depends on that of the previous run. This graph assumes that the main memory database buffer is large enough so that $p_{h2} = 1$, that is, no repeated I/Os occur during the rerun of a transaction for the OCC schemes. Therefore, the number of I/Os in a run for either the pure OCC or the OCC with broadcast during rerun schemes do not depend on the previous run, leading to better accuracy. For the transaction rate supportable for this case, the mean number of concurrent transactions (given by the product of the transaction rate and mean response time) is limited to about 40 transactions, each with a mean of 11 database I/Os, which (assuming a 4-Kbyte block of data for each I/O) corresponds to about 1.76 Mbytes of buffer memory. Since buffer memory of several times this size is feasible, the case in this graph of no repeated I/Os during transaction reruns is reasonable. A detailed buffer model appears in [11].

Comparing the mean response times for the different schemes, the response time for the locking scheme increases rapidly with increase in the transaction rate, due to long waiting times for locks for this high data contention case. The main reason that pure OCC outperforms broadcast OCC at high data contention is due to fewer aborts. This actually leads to lower CPU utilization under pure OCC in spite of the fact that the transaction is aborted earlier under broadcast OCC [41]. The reason for this striking behavior is we assumed that there is sufficient buffer memory so that a data block read does not need to be reread from disk again during subsequent reruns of the transaction.

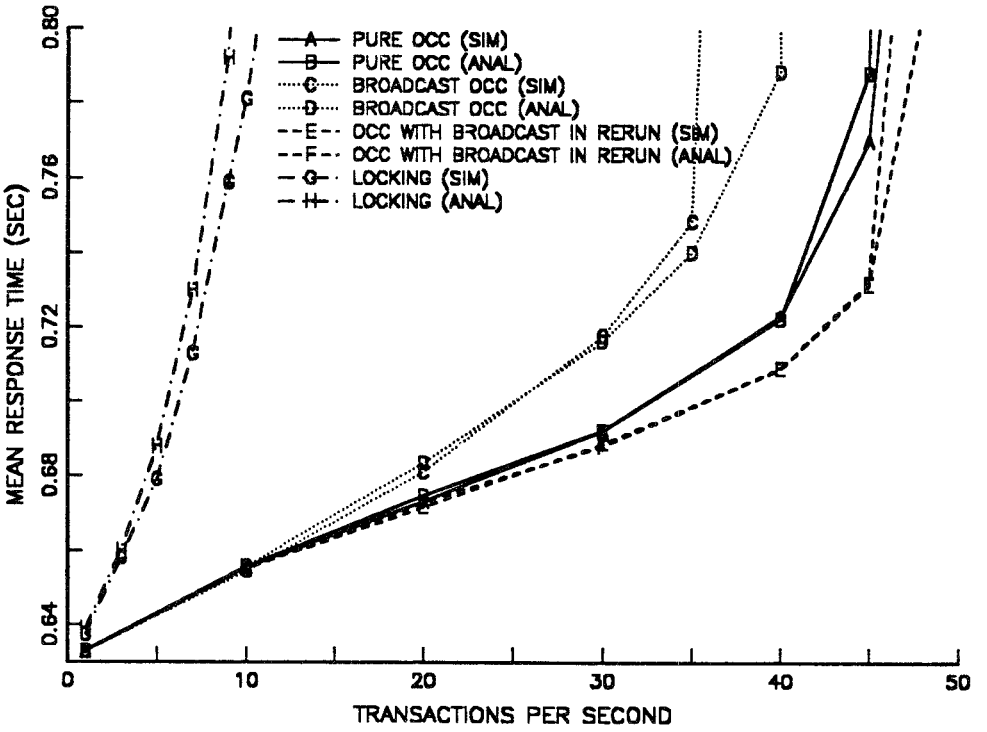

FIG. 4. Mean response time vs. throughput for different CC schemes.

Hence, the second and further runs are very short for pure OCC, but can be long for broadcast OCC since some I/Os are done in the reruns. This leads to a larger probability of second and subsequent aborts for broadcast OCC, and hence to larger CPU utilization. OCC with broadcast during rerun can reduce the CPU utilization by aborting the rerun transactions earlier and can thus improve performance over pure OCC in this case.

In Figure 5, we examine the accuracy of the analysis with variation in the level of data contention, by varying the number of granules in the database. For legibility, the figure shows the throughput–response time characteristics for only the locking and pure OCC schemes. The figure again shows good agreement between the approximate analysis and simulation estimates. Note that the CPU utilization is close to 100% beyond the knee of the curves for the pure OCC schemes. Comparing the curves, we note that locking shows vast improvement as the level of data contention reduces, that is, the number of granules increases, while the pure OCC scheme shows much less improvement. For a given transaction rate, the CPU utilization for locking is always lower than that for OCC. With few granules like 1K and 5K, the performance of locking is data contention limited, not CPU limited. When the number of granules increases to the point where data contention is low enough so that the CPU begins to become the limiting factor, the performance of locking starts to surpass pure OCC as the transaction rate increases. At lower data contention levels, the difference between the various OCC schemes also decreases.

In order to examine the application of the analysis to cases with a much higher transaction rate, Figure 6 shows the throughput–response time charac-

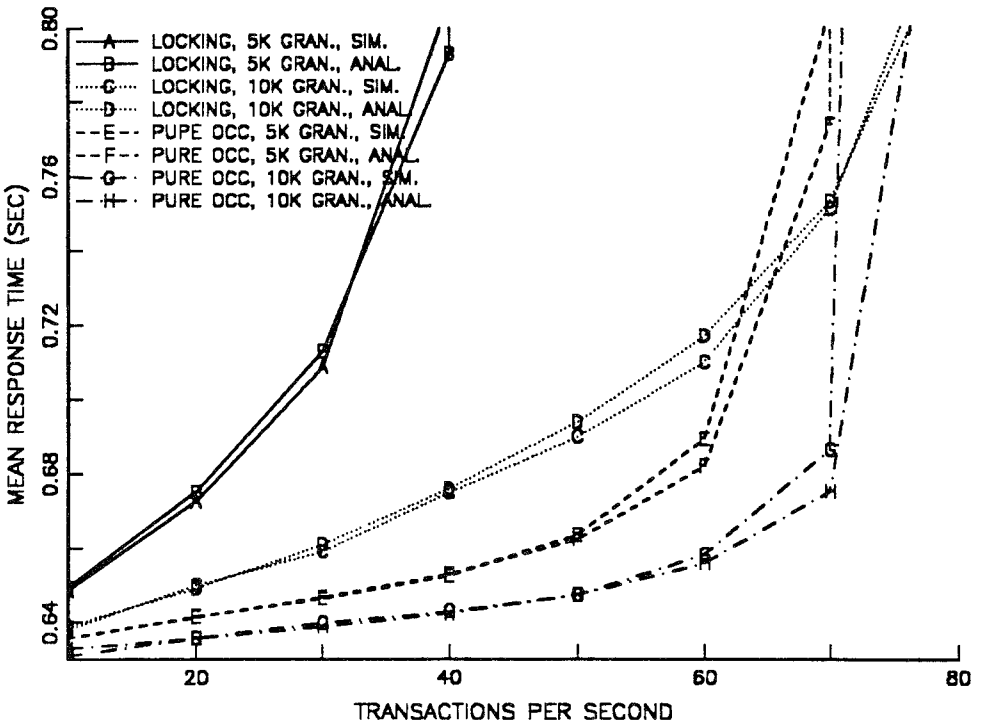

FIG. 5. Sensitivity to database size.

teristics for the case of 20,000 granules and 5 tightly coupled processors of 100 MIPS each. The rest of the parameters are the same as those for Figure 4. This was the largest number of granules that we were able to simulate successfully, and the processor MIPS is increased in order to support larger transaction rates. The difficulty in simulating such cases with large numbers of granules, in addition to long simulation times, increases the importance of the analysis. As before, there is good agreement between the estimates from the simulation and the analysis. These parameters again lead to a high level of data contention, and therefore the throughput for locking is data contention limited. There is some discrepancy between the simulation and analytical response time estimates for the OCC schemes close to the limiting throughput. This discrepancy is at high levels of CPU utilization. For instance, for the pure OCC scheme, the CPU utilization is about 90% at 600 transactions per second.

We now examine the effect of variable-length transactions on the system performance. We consider the case with the same parameters as that for Figure 5 with a database of 10K granules, except that there are two classes of transactions: one class of transactions, comprising 60% of the transactions, accesses 5 granules, while the second class, comprising the remaining 40%, accesses 30 granules. The instructions executed in each state and the mean number of I/Os per state are the same as those above. These parameters are chosen such that the mean number of granules accessed by transactions in the absence of conflict (i.e., in the first run of transactions) is the same (15 per transaction) for the fixed- and variable-length cases considered. This maintains the same mean number of instructions, mean number of I/Os, and mean locks

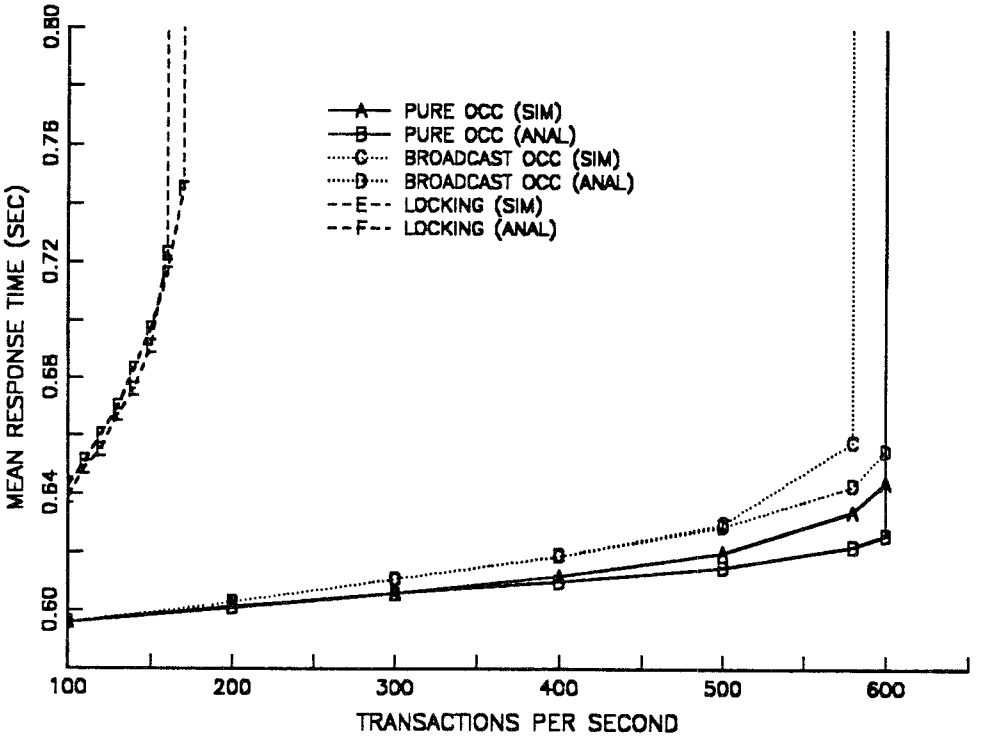

FIG. 6. Effect of increased MIPS and database size.

per transaction in the absence of conflict, allowing for an equitable comparison between these two cases.

The mean response time versus throughput characteristic for this case is shown in Figure 7. Comparing Figures 5 and 7, we observe that the change to variable transaction length reduces the maximum throughput that can be sustained by the locking and pure OCC protocols. However, for this choice of parameters, the impact of the increase in the variance of the transaction length on the maximum throughput that can be sustained is much larger for the locking protocol than for the optimistic protocols. The cause of this effect can be seen from Figures 8 through 10. Figure 8 shows the probability of the first abort of a transaction versus the transaction rate for the same parameters as Figure 7 for the pure OCC protocol. For the variable-length transaction case, the probabilities of abort for the short and long transactions are shown separately, and the overall transaction abort probability is also indicated. Note the large spread in the probability of abort for the different transaction lengths. Note also the close agreement of the simulation and analysis for the overall transaction abort probability. For comparison, the probability of abort for the corresponding case of fixed-length transactions is also shown in the same figure. Note that the increase in the overall transaction (first) abort probability due to the variable length is small. However, since the aborts are largely of the long transactions, and the long transactions consume more resources in the rerun than do the fixed-length transactions, the CPU utilization is larger for the variable-length transaction case, and it is this effect on

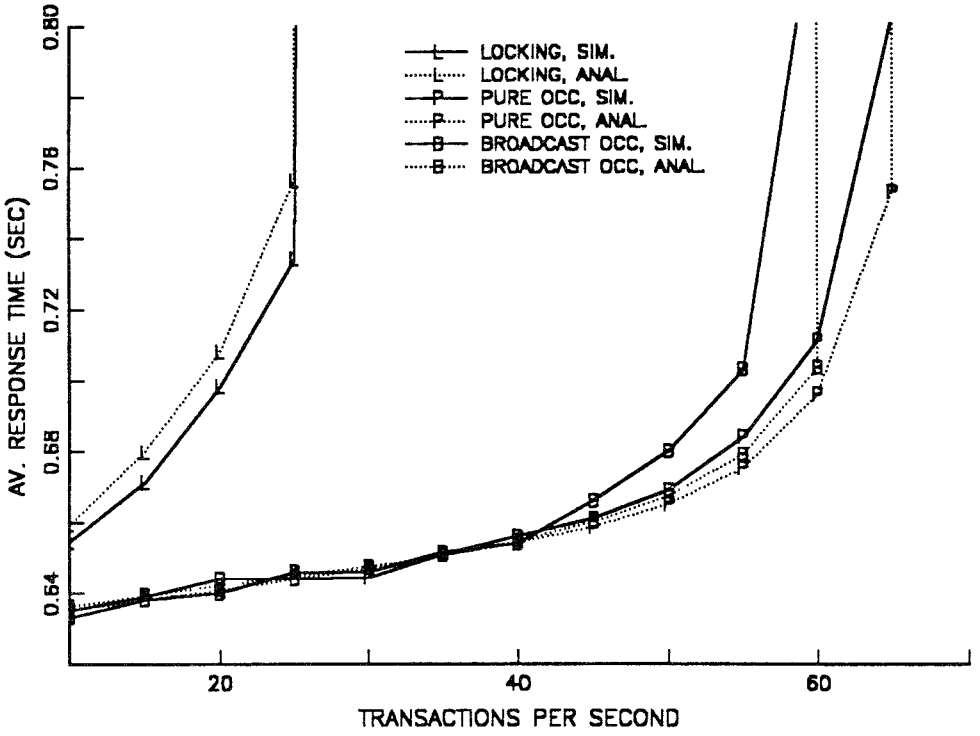

FIG. 7. Effect of variable-length transactions.

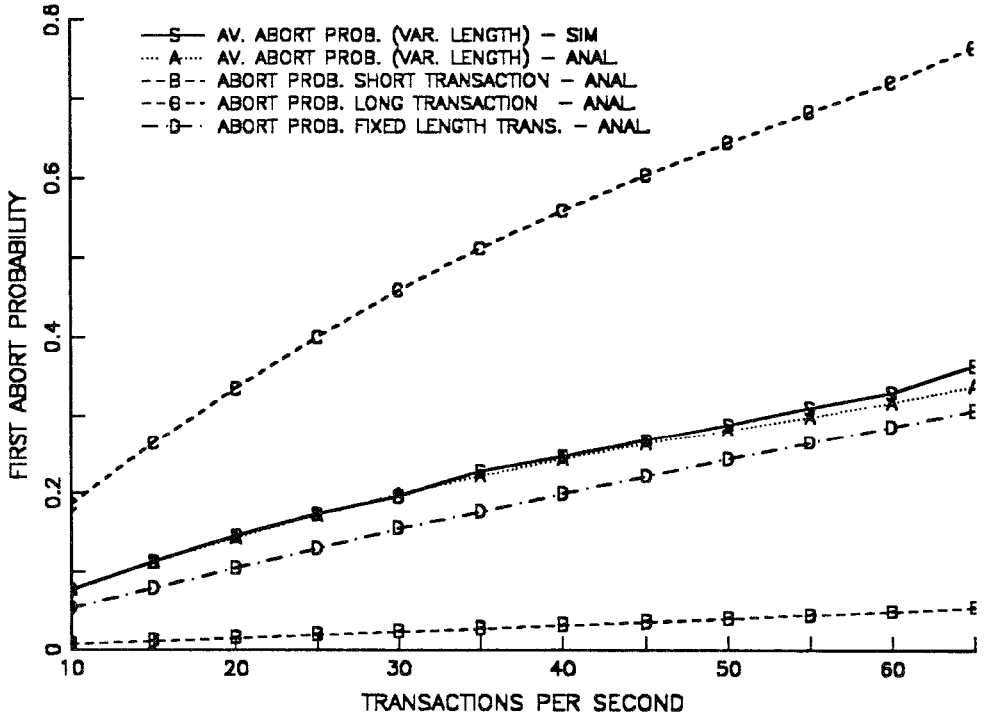

FIG. 8. First abort probability of Pure OCC with variable-length transactions.

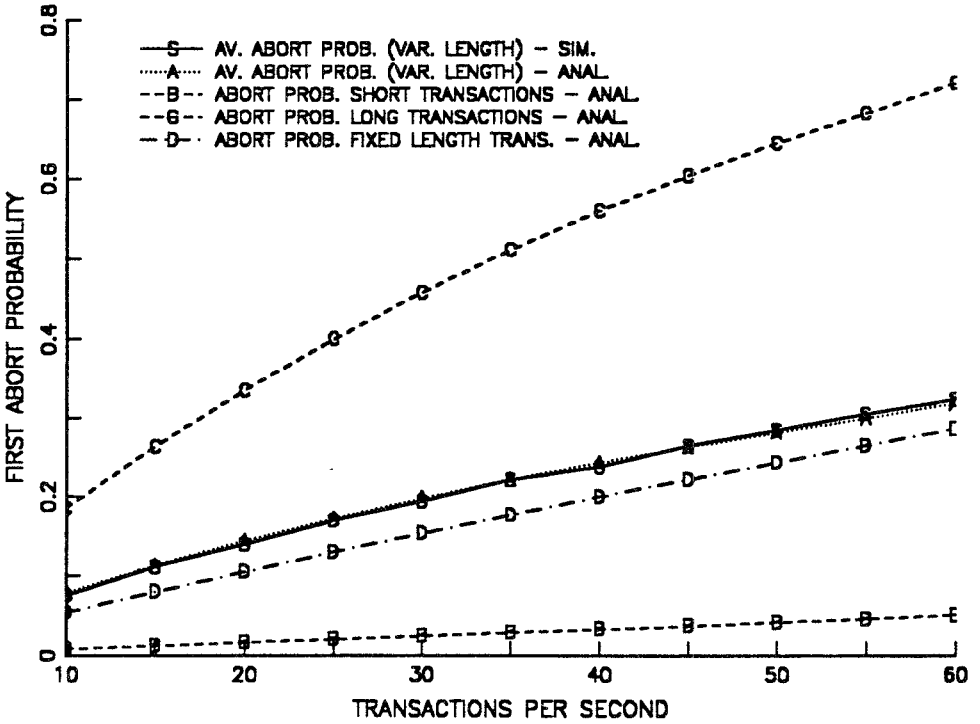

FIG. 9. First abort probability of Broadcast OCC with variable-length transactions.

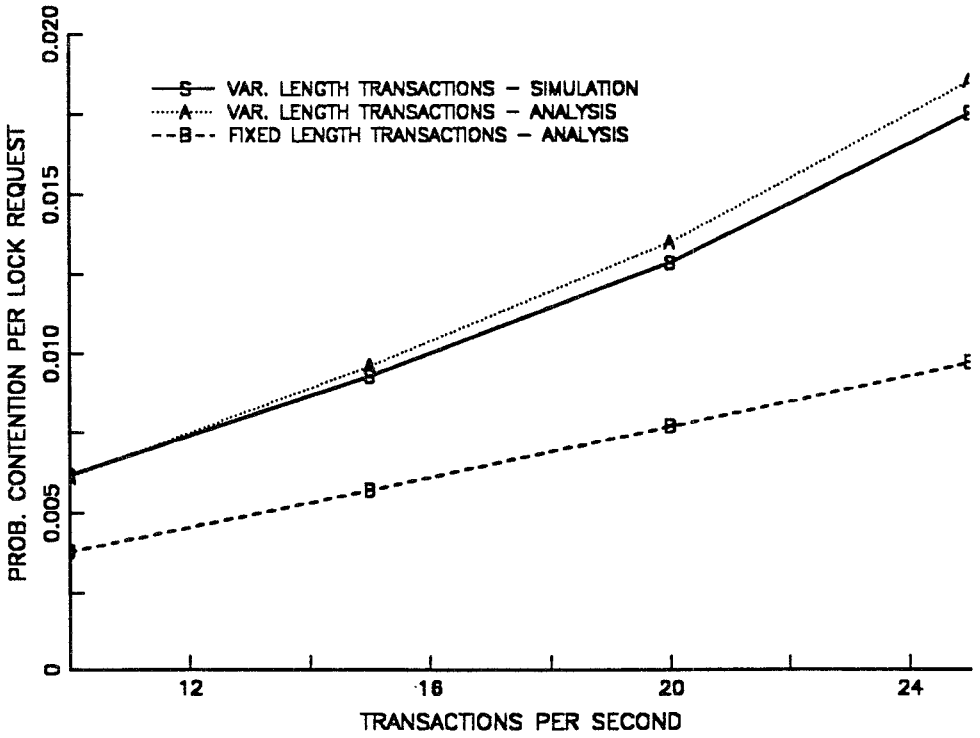

FIG. 10. Lock contention probability for 2PL with variable-length transactions.

CPU utilization that is mainly responsible for the reduction in the maximum throughput of the pure OCC scheme. Figure 9 is similar to Figure 8, and is for the broadcast OCC protocol. Again, the aborts are skewed towards longer transactions, increasing CPU utilization. Further, the longer transactions that are aborted have a larger probability of being aborted in the rerun than they do for the pure OCC protocol, leading to a further increase in the CPU utilization. Therefore, the effect of variable-length transactions is larger on the broadcast OCC protocol than on the pure OCC protocol. Finally, Figure 10 shows the probability of contention per lock request for the locking scheme for both the variable and the corresponding fixed-length transaction cases. Notice that the data contention probability for variable-length transactions is significantly larger than that for fixed-length transactions. The increased data contention in the locking protocol is largely due to the increase in the mean waiting time. This is because the probability of contention with a particular transaction is directly proportional to the number of locks held by the transaction. Therefore, transactions are more likely to contend with long transactions that hold a larger number of locks, and consequently, the mean lock waiting time is larger, leading to a further increase in the data contention probability. Now, for the above parameters with 10K granules in the database, the maximum throughput that can be sustained by the locking protocol becomes limited by data contention, and therefore the impact of the variable length is dramatic.

3.4. Fixed Multi-Programming Level. A number of studies in the literature (e.g., [35] and [36]) have assumed a fixed Multi-Programming Level (MPL) where a new transaction enters the system when one leaves. The following iterative approach can be used to approximately analyze the resulting models. The development in Sections 3.1 and 3.2 approximates the mean response time, given a transaction rate. Given an MPL, we start with an assumed (low value) of transaction rate (λ), estimate the mean response time (R), and then apply Little's law ($\lambda = \text{MPL}/R$) to estimate a new transaction rate. The results of such an iteration, for the same parameter values of Section 3.3 with 10K granules and fixed-length transactions are shown in Figure 11 for the locking and pure OCC schemes. The figure shows good agreement between simulation and analysis up to the point where the throughput peaks. Beyond this point the analysis does not converge (and this is easily detected by checking for convergence after a few (e.g., 50) iterations). The region beyond this has been referred to as the thrashing region [35]. Since operation in the thrashing region should be avoided, projecting response times in this region is not usually required. (Our simulations showed a large variance in the response time estimates in this region.) The throughput peaks for different reasons for locking and pure OCC. For locking, it peaks due to long lock waiting times, and for pure OCC it peaks due to high CPU utilization caused by transaction reruns. (The fact that the throughputs for locking and pure OCC in Figure 11 peak at about the same MPL and achieve about the same maximum value is coincidental for the parameter values used and will not be the case in general.) Comparing Figures 5 and 11 for the case of 10K granules shows that the maximum throughput obtained in the open model is about the same as that for the case with fixed MPL. For locking, as derived from Eq. (3.24), the mean number of contentions per transaction at which the maximum throughput is obtained for the open model is $P_{\text{W\_MAX}}^{T} = 0.75$. This value of the contentions

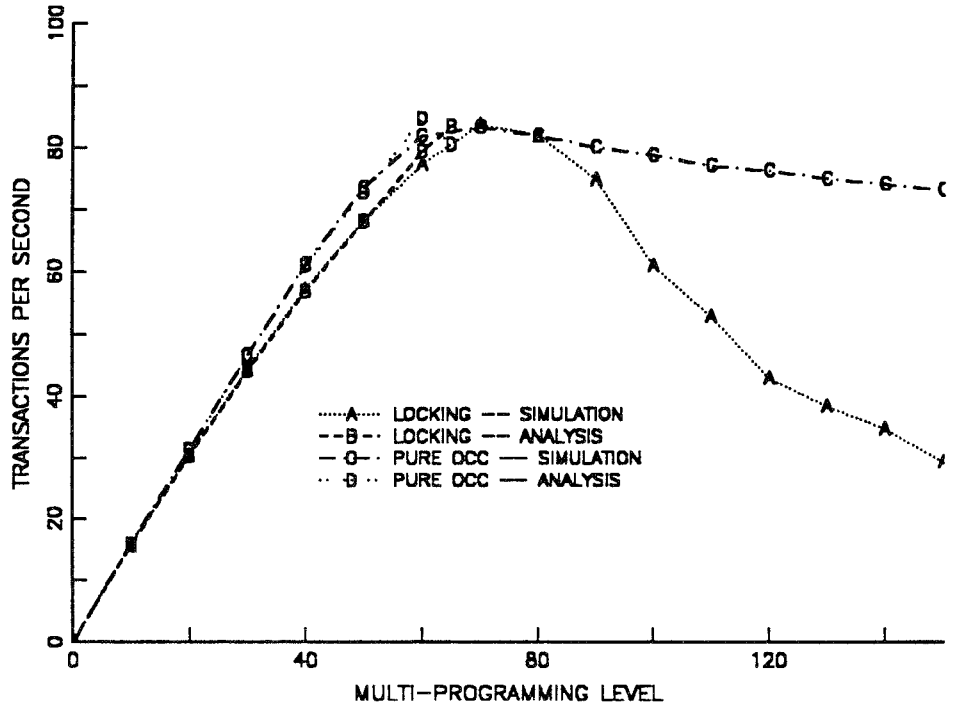

FIG. 11. Fixed multi-programming level.

per transaction at maximum throughput has also been reported in [24], [35], and [40]. The MPL required to stay away from the thrashing region can also be estimated from the characteristics of the open model by applying Little's law. For instance, Figure 5 shows that at the knee of the characteristic for locking with 10K granules, a mean response time of about 0.79 seconds is obtained for a throughput of 80 transactions per second, corresponding to an average MPL of about 63, which is close to the thrashing region in Figure 11.

4. *Application to Distributed Systems*

To further illustrate the application of our methodology, we next consider how to approximate the mean response time in distributed database systems. Replication of data can be used to reduce the amount of communication. Systems with fully replicated data have been studied analytically (e.g., [19] and [27]). With the exception of [9] which is closely related to our work here, systems with partially replicated data have only been studied via simulation (e.g., [1], [4], and [37]). We believe that the analysis we present here clarifies the analysis in [9]. We consider partial replication in this section. To maintain coherency among the replicates, one strategy is to have one copy designated as the primary copy [2]. The CC manager at the site of the primary copy is the global CC manager of that database. Transaction commits and granule updates are coordinated through the global CC manager. This is the strategy we consider.

4.1. Effect of Communications Delay. In a distributed system, there can be many databases. Each may have multiple copies scattered over different sites. In [9], it is shown that an optimistic type CC for intersite concurrency control leads to better performance than the pessimistic locking scheme. Hence, we focus on a generic OCC scheme to maintain intersite concurrency control. We first want to show the effect of communications delay on the abort probability. We start by examining the simple situation where there is just one transaction type, one database, and two sites, each of which has a copy of the database and where one is the primary site. We also assume that the data access distribution is uniform and that all accesses are exclusive. For the moment, we assume that the commit process at the primary site is instantaneous as in Section 2 so that the effect of communications delay is clear. Later, we examine the effect of nonzero commit time. The global CC manager maintains a table to track the version of each data granule. At the commit point, a transaction at the nonprimary site communicates to the primary site to check whether all data accesses are to the up-to-date version; if so, it commits; otherwise, it is aborted. All granules updated by committed transactions are conveyed to the nonprimary site after the commit point. This is referred to as asynchronous update in contrast to synchronous update where updating replicates is done in the commit interval.

Let λ be the total transaction rate which is equally split between the two sites, $\hat{R}$ be the mean execution time in state i for a transaction whether running at the primary site or nonprimary site, and D_C be the mean communications delay of a message between the sites. Assume that a transaction x running at the nonprimary site accesses a fixed number N_L of granules. (Generalization to variable length is similar to that given in Section 3.) We first consider the effect of communications delay on abort probability. The communications delay increases the abort probability of transactions running at the nonprimary site in two ways. First of all, a granule updated by a transaction running at the primary site is not reflected at the nonprimary site until a mean time of D_C after the commit point of the primary-site transaction. Therefore, a granule accessed by a transaction running at the nonprimary site can get invalidated due to primary-site commits that occur during an interval of mean duration D_C before the granule is actually accessed by the nonprimary-site transaction. Secondly, the commit request of a nonprimary-site transaction arrives at the primary site after a mean delay of D_C. Any conflicting transactions at the primary site requesting commit during this interval will cause the nonprimary-site transaction to abort. Essentially, the mean exposure window of each granule of the nonprimary transaction is increased by an additional D_C by each of these two factors. This is equivalent to increasing the effective mean holding time on each granule access by $2D_c$. Thus, similar to the derivation of Eq. (2.10), the abort probability of the nonprimary-site transactions can be expressed as,

$$P_A^{NP} \simeq 1 - \exp\left(-\frac{\lambda(T_H + 2D_C)N_L^2}{L}\right), \tag{4.1}$$

which can be further approximated as,

$$P_A^{NP} \simeq \frac{\lambda(T_H + 2D_C)N_L^2}{L}. \tag{4.2}$$

(Note that T_H is the mean granule holding time until the commit point, excluding the time to communicate to the primary site.) Since no communication is required for transactions that run at the primary site, the abort probability, P_A^p, of primary-site transactions can be estimated using Eq. (2.10).

4.2. REPLICATED DATABASES. We next examine how the analysis can be generalized to an environment with many databases and many sites. A transaction executes completely at the site to which it arrives and accesses databases for which (a) its site has the primary copy, (b) its site has a nonprimary copy, or (c) its site has no copy. In case (b), the data is obtained from the local nonprimary copy, and in case (c) it is obtained from the remote primary copy. In all cases, global concurrency control and commit go through the global CC manager at the primary-copy site as explained later.

In [9] several concurrency control schemes are considered to study the effects that various degrees of replicating data in a distributed database environment have on different CC schemes. The CC schemes examined include locking, pure OCC, and a semi-optimistic (SOCC) scheme. Here we focus on how to analyze the SOCC scheme that combines locking and OCC. The specific SOCC scheme considered works as follows: The global CC manager at the primary-copy site maintains a lock table. Transactions accessing a database whose primary copy is local (case (a) above) obtain a lock before each granule access. Transactions accessing a database for which there is a local nonprimary copy (case (b) above) try to obtain locks at commit time for all granules accessed. In case (b), the transaction only communicates with the primary-copy site at commit time. It requests the global CC manager to certify its granule accesses and accept its updates. In case (c), where there is no local copy, communication with the primary-copy site is of course required to access the granules and in addition communication is required at commit time to certify that its granule accesses are still valid and have its updates accepted. The global CC manager for a database not only checks for whether the locks can be granted for granules accessed, but also verifies whether the granules accessed are up to date. If locks for all the granules accessed cannot be obtained or the granules accessed are obsolete, the transaction is aborted. This is basically the pure OCC approach. Otherwise, the global CC manager grants the locks and permits the transaction to enter a second commit phase [13], and a positive acknowledgment is sent back to the transaction's site. If all sites involved returned positive acknowledgments, the transaction will be committed, and update messages will be sent to all sites with a replicate of the database. In the following analysis, we assume asynchronous updates to the replicates. The synchronous case can be similarly analyzed. Note that the existence of a local replicate provides local accesses to the granules but does not eliminate the need to communicate with the global CC manager to validate the accesses and commit the updates. Each replicate still needs a local CC manager to track the version of each data granule accessed. At the end of the execution phase, if obsolete data accesses are detected by the local CC manager, the transaction will be aborted without even communicating to the global CC manager. Otherwise, the local CC manager will make the commit request to the global CC manager on behalf of the transaction.

We assume two transaction classes. Class 1 transactions only access granules from databases whose primary copy is local. Class 2 transactions access some

granules from databases with a remote primary-copy site, where a local replicate may or may not exist. All granule accesses are assumed to be exclusive, and the access distribution within a database is assumed to be uniform. Let us examine a specific database j with L_j granules. Let λ_1^j be the arrival rate of class 1 transactions that access database j and λ_2^j be the arrival rate of class 2 transactions that access database j. Let $T_{H_1}^j$ be the mean granule holding time for the class 1 transactions accessing database j, $T_{H_2}^j$ be the mean granule holding time for the class 2 transactions accessing database j (excluding the commit period), and T_{C_2} be the mean commit time for the class 2 transactions. Let $N_{L_1}^j$ be the fixed number of granules accessed by class 1 transactions from database j and $N_{L_2}^j$ be the fixed number accessed by class 2 transactions from database j. Furthermore, let $\hat{R}^j$ be the average time spent between the ith and the $(i+1)$th access to granules in database j by a class 2 transaction, which is assumed to be independent of i. Then $T_{H_2}^j = (N_{L_2}^j + 1)\hat{R}^j/2$. In a similar manner to that used to obtain the approximation in Eq. (3.1) and with the communication delay handled as in Eq. (4.1), the abort probability of class 2 transactions due to accesses to database j can be approximated as

$$P_{A_2}^j \simeq 1 - \exp\left(-\frac{\left(\left(\lambda_1^j N_{L_1}^j + \lambda_2^j N_{L_2}^j\right)\left(T_{H_2}^j + 2D_C\right)\right)N_{L_2}^j}{L_j}\right) \times \left(1 - \frac{\left(\lambda_1^j N_{L_1}^j T_{H_1}^j + \lambda_2^j N_{L_2}^j T_{C_2}\right)}{L_j}\right)^{N_{L_2}^j}, \tag{4.3}$$

which can be further approximated as,

$$P_{A_2}^j \simeq \frac{\left(\left(\lambda_1^j N_{L_1}^j + \lambda_2^j N_{L_2}^j\right)\left(T_{H_2}^j + 2D_C\right) + \lambda_1^j N_{L_1}^j T_{H_1}^j + \lambda_2^j N_{L_2}^j T_{C_2}\right)N_{L_2}^j}{L_j}. \tag{4.4}$$

There are two causes of abort reflected in Eq. (4.3), one due to invalidation, reflected in the exponential expression, and one due to lock contention at commit time. In the exponential expression in Eq. (4.3), $(\lambda_1^j N_{L_1}^j + \lambda_2^j N_{L_2}^j)L_j$ is the invalidation rate for one of the $N_{L_2}^j$ granules accessed by a class 2 transaction from database j and $(T_{H_2}^j + 2D_C)$ is the mean equivalent "effective" database j granule holding time for a class 2 transaction as explained before. In the expression that postmultiplies the exponential expression, $(\lambda_1^j N_{L_1}^j T_{H_1}^j + \lambda_2^j N_{L_2}^j T_{C_2})/L_j$ is the total lock utilization of one of the $N_{L_2}^j$ locks requested at commit time by a class 2 transaction. The approximation in Eq. (4.4) can be obtained by series expansion of the expressions in Eq. (4.3). However, it can also be written down directly using the additive approximation discussed in Section 2.2.

Assume that class 2 transactions access a fixed number m of different databases. The overall abort probability for class 2 transactions can be approximated as $1 - \prod_{j=1}^{m}(1 - P_{A_2}^j)$, which is approximately $\sum_{j=1}^{m} P_{A_2}^j$.

As just discussed, the utilization for a lock on a granule in database j is

$$\frac{\left(\lambda_1^j N_{L_1}^j T_{H_1}^j + \lambda_2^j N_{L_2}^j T_{c2}\right)}{L_j}.$$

Then, making the same assumptions as in Proposition 2.1, this also equals the lock contention probability for class 1 transactions at the primary site of database j. After obtaining the conflict probabilities, we can proceed to couple these with the hardware resource model as before and approximately analyze the performance.

The locking and pure OCC schemes can be similarly analyzed. Consider the pure OCC case where the global CC manager at the primary site uses the pure OCC scheme as described in Section 3.1 for all transaction accesses. Divide T_{H_1} into two parts: $T^j_{C_1}$ the commit interval for class 1 transactions, and $T^j_{H_1'}$, the mean granule holding time excluding the commit interval. The abort probability of class 2 transactions due to accesses to database j under pure OCC is the same as $P^j_{A_2}$ in Eq. (4.3) or (4.4) with the exception that $T^j_{H_1}$ needs to be replaced by $T^j_{C_1}$. The abort probability of class 1 transactions under pure OCC is the same as that of class 2 transactions except the term $(T^j_{H_2} + 2D_C)$ is replaced by $T^j_{H_1'}$ in the numerator and the last term in the numerator, $N^j_{L_2}$, is replaced by $N^j_{L_1}$. Finally, we consider the locking case where the global CC manager at the primary site uses the standard locking scheme for all transactions. The lock contention probability due to an access to database j for either class is $(\lambda^j_1 N^j_{L_1} T^j_{H_1} + \lambda^j_2 N^j_{L_2}(T^j_{H_2} + T_{C_2} + 2D_C))/L_j$. Careful checking shows that the conservation property for the overall conflict probability still holds for all these cases. More details can be found in [9], where the analysis further distinguishes between rerun transactions and first-run transactions as in Section 3.1.

In [9], comparisons are given of the approximate analytical results with simulation results for $O(1/L)$ approximations similar to that in Eq. (4.4). The comparisons are for a 10-site distributed system using the SOCC scheme with full and partial replication. The simulation and analytical results for the mean response times of class 1 and class 2 transactions are quite close.

5. *Summary and Conclusion*

The design and analysis of the CC scheme can be crucial to the successful operation of a transaction-processing system. However, the complexity of the system makes it difficult to understand the trade-offs between the different CC schemes in a particular operating environment without detailed analysis. Simulation models of transaction-processing systems can often be very time consuming to run. As mentioned in the Appendix, the simulations required to generate each curve in the paper typically took around 1 hour of CPU time on a large IBM mainframe to obtain good statistical accuracy. Therefore, for exploring a wide variety of design alternatives, it would be useful to have a unified analytical methodology. In this paper, we have developed a unified approximate analytical methodology to study the effects of different CC schemes over a spectrum of environments including centralized and distributed transaction-processing systems with optimistic and pessimistic type CC schemes.

The methodology decomposes the model into submodels for hardware resource contention and for data contention. We focus on the data contention model, and as others have done, we iterate between the solutions of the data contention model and a standard hardware resource contention model to obtain the overall system performance. For data contention, the analysis proceeds based on probabilistic assumptions that are explicitly stated. In

particular, we assumed for locking schemes that the lock request times form a Poisson process and that lock contention events for a transaction are independent and occur with the same probability and for OCC schemes that the invalidation times for each granule form a Poisson process and that the processes for different granules are independent. These assumptions allowed us to obtain simple approximations for the probability that a transaction encounters lock contention when locking is used, and for the probability that a transaction is aborted when OCC is used. The approximations were obtained for idealized versions of locking and OCC in Section 2 and then extended to more realistic versions for centralized systems in Section 3 and for distributed systems in Section 4. We observed in Section 2.2 that for the idealized CC schemes the dominant, $O(1/L)$, terms in the probability of conflict approximations are additive in the various sources of conflict. This observation provides a method for writing down by inspection the dominant terms in the approximations for the conflict probabilities for a variety of more realistic CC schemes. (These approximations can also be derived as was done in this paper.) Furthermore, it is often the case that different CC schemes have the same approximate expression for the overall conflict probability since the same sources of conflict are present. This was called the *conservation property* of conflict. (Cases where the conservation property does not hold were also mentioned.) However, the CC schemes profoundly affect the mean granule holding time that appears in the expressions and therefore lead to very different performance.

The results of the approximate analysis were validated by comparison with simulation results in a wide variety of cases for centralized transaction-processing systems. Overall, the accuracy is good, and we believe that the methodology presented here is useful in obtaining insights into the performance implications of different CC schemes and system structures without resorting to time-consuming simulations. For example, for OCC schemes in centralized systems, we examined the effect of a large buffer that results in rerun transactions finding most of the granules in the buffer. For this case we showed that, at high contention levels, pure OCC that aborts transactions at commit can actually be superior to broadcast OCC that attempts to abort a transaction as soon as a conflict is detected. We also compared the performance of locking with that of the OCC schemes. To further illustrate our methodology, we showed how it can be applied in the distributed and replicated data case.

Appendix A

In this appendix, we outline the simulation model mentioned in Section 3 and discuss the confidence intervals that were obtained from the simulation runs. All of the CC schemes analyzed therein were simulated using a discrete-event simulation model. The simulation keeps track of the data accessed by each transaction and explicitly simulates data contention, transaction aborts, locking of data granules, waiting for locks to become available, queuing and processing at the CPU, I/O waits, communication delays, and commit processing. The tightly coupled processors are simulated as having a common work queue, while geographically distributed processors have separate queues. All transaction arrival processes are independent and Poisson. The CPU service times are constants that correspond to the CPU MIPS rating and the specific instruction

pathlengths given in Section 3.3. (They are not exponentially distributed as in the analytical model.) Communication delays, commit times, I/O times, and backoff times are all constants. The CPU is released by a transaction when lock contention occurs, for each I/O and during backoff after an abort.

The states of a transaction in the simulation are the same as described in Section 2 for the analysis. An incoming transaction is first queued at the CPU for the initial setup processing, interspersed with the initial I/Os. At the start of each subsequent state, a request is made to the CC component. For the locking scheme, the lock table is checked, and if a contention is detected the CPU is released, and the transaction is enqueued on that lock. For the broadcast OCC scheme, a check is made if the transaction has been marked for abort, and if so a backoff interval is started. The phase of the transaction (see 3.1.2) is also updated. For the pure OCC scheme and if no aborts are detected in the broadcast OCC scheme, the transaction identifier is entered in an access table, which is a list of transactions accessing each granule. Following this, the transaction is queued at the CPU for processing followed by an I/O with probability $(1 - p_{h1})$ in the the first run and probability $(1 - p_{h2})$ in subsequent runs if the granule has already been read in the previous runs. An infinite server model is used for the I/O, and I/O times are constant. In the simulation, granule accesses from different transactions are independent with replacement, while accesses from the same transaction are independent without replacement. (In the analysis, we assumed independence with replacement for accesses from the same transaction.) In addition, in the simulation, rerun transactions access the same granules accessed in the first run of that transaction. In the case of a contention that leads to a deadlock, the transaction that just made the lock request is aborted, all locks held are released and a backoff interval is started. At commit point for the OCC schemes, if the transaction is marked for abort, a backoff interval is started; otherwise, any transactions accessing granules in incompatible mode are marked for abort, and locks are granted on the granules previously accessed by the committing transaction.

In the simulations results reported, confidence intervals were obtained (but not shown on the graphs) using the method of batch means [26]. In the simulations, the statistics of the first 2,000 transactions were discarded, followed by gathering statistics for 50,000 transactions, divided into 25 batches of 2,000 transactions each. Assuming independence of the estimates derived from each batch, confidence intervals were generated. The simulation run time for each case depends on the parameters, with longer simulation times for cases with larger contention and abort probabilities, but each curve shown takes approximately between 45 minutes and 1.5 hours of CPU time on an IBM 3090 (with, of course, significantly longer wall time), thus establishing the value of the analysis in getting quick and accurate estimates. For most cases, the estimated confidence intervals are very tight, except at very high data contention levels or CPU utilizations. Illustrative examples are as follows. In Figure 1 at 10 transactions per second (tps) the 95% confidence interval of the transaction abort probability is estimated as within ± 0.004 of the point estimate (shown in the figure), while at 45 tps it is within ± 0.007 of the point estimate, so that the confidence interval half widths are less than 1% of the point estimates. Similar results were obtained for Figure 2. In Figure 3, the 95% confidence interval is within 3% of the point estimate at 10 tps, and uniformly tighter at lower transaction rates. In Figures 4 and 5, the 95%

confidence intervals are within 1% of the point estimates for all the curves and data points shown. The point estimates in the variable-length transaction case in Figures 6 through 9 have a larger variance leading to 95% confidence intervals within 5% of the point estimates. In Figure 10, the 95% confidence interval for the pure OCC case is consistently within 1% of the point estimates shown. For the locking case, the 95% confidence interval is within 3% of the point estimates for other than the thrashing region (MPL less than 70); in the thrashing region, the corresponding confidence interval widens, increasing up to $\pm 5\%$ of the point estimate shown for an MPL of 130.

REFERENCES

1. BARBARA, D., AND GARCIA-MOLINA, H. How expensive is data replication? An example. In *Proceedings of the 2nd International Conference on Distributed Computing Systems* (Feb.) IEEE Computer Society Press, Los Alamitos, Calif., 1982, pp. 263–268
2. BERNSTEIN, P. A., HADZILACOS, V., AND GOODMAN, N. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, Reading, Mass., 1987
3. BOKSENBAUM, C., CART, M., FERRIE, J., AND PONS, J.-F. Concurrent certifications by intervals of timestamps in distributed database systems. *IEEE Trans. Softw. Eng. SE-13*, 4 (Apr. 1987), 409–419.
4. CAREY, M. J., AND LIVNY, M Distributed concurrency control performance: A study of algorithms, distribution, and replication. In *Proceedings of the 14th Annual Conference on Very Large Data Bases*. Morgan Kaufmann, San Mateo, Calif., 1988, pp 13–25.
5. CELLARY, W., GELENBE, E., AND TADEUSZ, M. *Concurrency Control in Distributed Database Systems*. North-Holland, Amsterdam, The Netherlands, 1988.
6. CHESNAIS, A., GELENBE, E., AND MITRANI, I. On the modeling of parallel access to shared data. *Commun. ACM 26*, 3 (Mar. 1983), 196–202.
7. CICIANI, B., DIAS, D. M., IYER, B. R , AND YU, P. S. A hybrid distributed centralized system structure for transaction processing. *IEEE Trans. Softw. Eng. SE-16*, 8 (1990), 791–806.
8. CICIANI, B., DIAS, D. M., AND YU, P. S. Analysis of concurrency-coherency control protocols for distributed transaction processing systems with regional locality. *IEEE Trans. Softw. Eng. SE-18*, 10 (Oct. 1992), 889–914.
9. CICIANI, B., DIAS, D. M., AND YU, P. S. Analysis of replication in distributed database systems. *IEEE Trans. Knowledge Data Eng. 2*, 2 (June 1990), 247–261.
10. CORNELL, D. W., DIAS, D. M , AND YU, P. S. On multisystem coupling through function request shipping. *IEEE Trans. Softw. Eng. SE-12*, 10 (Oct. 1986), 1006–1017.
11. DAN, A., DIAS, D. M., AND YU, P. S. The effect of skewed data access on buffer hits and data contention in a data sharing environment. In *Proceedings of the 16th International Conference on Very Large Databases* (Brisbane, Australia, Aug.). Morgan Kaufmann, San Mateo, Calif , 1990, pp. 419–431.
12. DAN, A., TOWSLEY, D. F., AND KOHLER, W. H. Modeling the effects of data and resource contention on the performance of optimistic concurrency control protocols. In *Proceedings of the 4th International Conference on Data Engineering* (Los Angeles, Calif., Feb.). IEEE Computer Society Press, Los Alamitos, Calif., 1988, pp. 418–425.
13. DATE, C. J. *An Introduction to Database Systems*. Vol. 2. Addison-Wesley, Reading, Mass., 1983.
14. DIAS, D. M., IYER, B. R, AND YU, P. S. Trade-offs between coupling small and large processors for transaction processing. *IEEE Trans. Comput. 37*, 3 (Mar. 1988), 310–320.
15. DIAS, D. M., YU, P. S , AND BENNETT, B. T. On centralized versus geographically distributed database systems. In *Proceedings of the 7th International Conference on Distributed Computing Systems* (Berlin, West Germany, Sept.). IEEE Computer Society Press, Los Alamitos, Calif., 1987, pp. 64–71.
16. FRANASZEK, P. A., AND ROBINSON, J. T Limitations of concurrency in transactions processing. *ACM Trans. Datab. Syst. 10*, 1 (Mar. 1985), 1–28.
17. FRANASZEK, P. A., ROBINSON, J. T., AND THOMASIAN, A. Access invariance and its use in high contention environments. In *Proceedings of the 6th International Conference on Data Engineering* (Los Angeles, Calif.). IEEE Computer Society Press, Los Alamitos, Calif., 1990, pp. 47–55.

18. FRANASZEK, P. A., ROBINSON, J. T., AND THOMASIAN, A. Wait depth limited concurrency control. In *Proceedings of the 7th International Conference on Data Engineering* (Kobe, Japan). IEEE Computer Society Press, Los Alamitos, Calif., 1991, pp. 92–101.
19. GARCIA-MOLINA, H. Performance of update algorithms for replicated data in a distributed database. Ph.D. dissertation. Computer Science Dept., Stanford Univ., Stanford, Calif., June 1979.
20. GRAY, J. N. An approach to decentralized computer systems. *IEEE Trans. Softw. Eng. SE-12*, 6 (June 1986), 684–692.
21. GRAY, J., HOMAN, P., OBERMARCK, R., AND KORTH, H. A straw man analysis of probability of waiting and deadlock. IBM Res. Rep. RJ 3066. IBM, Yorktown Heights, N.Y., 1981.
22. HARTZMAN, C. S. The delay due to dynamic two-phase locking. *IEEE Trans. Soft. Eng. 15*, 1 (Jan. 1989), 72–82.
23. HSU, M., AND SHANG, B. Modeling performance impact of hotspots. Tech. Rep. TR-08-88. Aiken Computation Laboratory, Harvard University, Cambridge, Mass., Apr. 1988.
24. IYER, B. R. Limits in transaction throughput—Why big is better. IBM Res. Rep. RJ 6584. IBM, Yorktown Heights, N.Y., Nov. 1988.
25. KUNG, H. T., AND ROBINSON, J. T. On optimistic methods for concurrency control. *ACM Trans. Datab. Syst. 6*, 2 (June 1981), 213–226.
26. LAVENBERG, S. S. (ED.) *Computer Performance Modeling Handbook*. Academic Press, Orlando, Fla., 1983.
27. MENASCE, D. A., AND NAKANISHI, T. Performance evaluation of a two-phase commit based protocol for DDBs. In *Proceedings of the ACM Symposium on Principles of Database Systems* (Los Angeles, Calif., Mar. 29–31). ACM, New York, 1982, pp. 247–255.
28. MORRIS, R. J. T., AND WONG, W. S. Performance analysis of locking and optimistic concurrency control algorithms. *Perf. Eval. 5* (1985), 105–118.
29. ROSENKRANTZ, D. J., STEARNS, R. E., AND LEWIS, P. M., II System level concurrency control for distributed database systems. *ACM Trans. Datab. Syst. 3*, 2 (June 1978), 178–198.
30. RYU, I. K., AND THOMASIAN, A. Performance analysis of centralized databases with optimistic concurrency control. *Perf. Eval. 7* (1987), 195–211.
31. SEVCIK, K. C. Comparison of concurrency control methods using analytic models. In *Information Processing 83*, R. E. A. Mason, ed. North-Holland, Amsterdam, The Netherlands, 1983, pp. 847–858.
32. SINGHAL, M., AND YESHA, Y. A polynomial algorithm for computation of the probability of conflicts in a database under arbitrary data access distribution. *Inf. Proc. Lett. 27*, 2 (Feb. 1988), 69–74.
33. TAY, Y. C. A mean value performance model for locking in databases. Ph.D. dissertation. Harvard University, Cambridge, Mass., Feb. 1984.
34. TAY, Y. C. Issues in modelling locking performance. In *Stochastic Analysis of Computer and Communication Systems*, H. Takagi, ed. North-Holland, Amsterdam, The Netherlands, 1990, pp. 631–655.
35. TAY, Y. C., GOODMAN, N., AND SURI, R. Locking performance in centralized databases. *ACM Trans. Datab. Syst. 10*, 4 (Dec. 1985), 415–462.
36. TAY, Y. C., SURI, R., AND GOODMAN, N. A mean value performance model for locking in databases: The no-waiting case. *J. ACM 32*, 3 (July 1985), 618–651.
37. THANOS, C., BERTINO, C., AND CARLESI, C. The effects of two-phase locking on the performance of a distributed database management system. *Perf. Eval. 8* (1988), 129–157.
38. THOMASIAN, A. An iterative solution to the queueing network model of a DBMS with dynamic locking. In *Proceedings of the 13th CMG Conference*. Computer Measurement Group, Alexandria, Va., 1982, pp. 252–261.
39. THOMASIAN, A., AND RYU, I. K. Analysis of some optimistic concurrency control schemes based on certification. *Perf. Eval. Rev. 13*, 2 (*Proceedings of 1985 ACM SIGMETRICS*), pp. 192–203.
40. THOMASIAN, A., AND RYU, I. K. Performance analysis of two-phase locking. *IEEE Trans. Softw. Eng. 17*, 5 (May 1991), 386–401.
41. YU, P. S., AND DIAS, D. M. Performance analysis of optimistic concurrency control schemes for systems with large memory. IBM Res. Rep. RC 13976. IBM, Yorktown Heights, N.Y., Mar. 1988.
42. YU, P. S., AND DIAS, D. M. Concurrency control using locking with deferred blocking. In *Proceedings of the 6th International Conference on Data Engineering* (Los Angeles, Calif.). IEEE Computer Society Press, Los Alamitos, Calif. 1990, pp. 30–36.

43. YU, P. S., AND DIAS, D. M. Analysis of hybrid concurrency control schemes for a high data contention environment. *IEEE Trans. Softw. Eng. 18*, 2 (Feb. 1992), 118–129.
44. YU, P. S., DIAS, D. M., CORNELL, D. W., AND IYER, B. R. Analysis of affinity based routing in multi-system data sharing. *Perf. Eval. 7*, 2 (June 1987), 87–109.
45. YU, P. S., DIAS, D. M., ROBINSON, J. T., IYER, B. R., AND CORNELL, D. W. Modelling of centralized concurrency control in a multi-system environment. *Perf. Eval. Rev. 13*, 2 (*Proceedings of 1985 ACM SIGMETRICS*), 183–191.
46. YU, P. S., DIAS, D. M., ROBINSON, J. T., IYER, B. R., AND CORNELL, D. W. On coupling multi-systems through data sharing. *Proc. IEEE 75*, 5 (May 1987), 573–587
47. YU, P. S., HEISS, H., AND DIAS, D. M. Modelling and analysis of a time-stamp history based certification protocol for concurrency control. *IEEE Trans. Knowl. Data Eng. 3*, 4 (Dec. 1991), 525–537.

RECEIVED JANUARY 1990; REVISED FEBRUARY 1991; ACCEPTED JANUARY 1992