Concurrency Control: Methods, Performance, and Analysis

ALEXANDER THOMASIAN

IBM T. J. Watson Research Center, 30 Saw Mill River Road, Hawthorne, NY 10532

Standard locking (two-phase locking with on-demand lock requests and blocking upon lock conflict) is the primary concurrency control (CC) method for centralized databases. The main source of performance degradation with standard locking is blocking, whereas transaction (txn) restarts to resolve deadlocks have a secondary effect on performance. We provide a performance analysis of standard locking that accurately estimates its performance degradation leading to thrashing. We next introduce two sets of methods to cope with its performance limitations. Restart-oriented locking methods selectively abort txns to increase the level of concurrency for active txns with respect to standard locking in high-contention environments. For example, the running-priority method aborts blocked txns based on the *essential blocking* principle, which only allows blocking by active txns. The wait-depth-limited (WDL) method further minimizes wasted processing by basing abort decisions on the progress made by a txn. Restart waiting serves as a load-control mechanism by deferring the restart of an aborted txn until conflicting txns have left the system. These two methods have performance superior to other restart-oriented methods and standard locking in high-contention environments. In two-phase processing methods an aborted txn may continue its first phase of execution in "virtual" mode, that is, without requesting any locks, prefetching data for its second execution phase. The second execution phase is shorter since no disk I/O is required, resulting in a lower effective degree of txn concurrency and less data contention. This method is effective provided *access invariance* prevails; that is, txns access the same set of objects in the second phase as they did in the first. The optimistic die method is appropriate for the first phase and the optimistic kill method for further phases. Lock preclaiming instead of the optimistic kill method in the second phase prevents further restarts, which is a weak point of the optimistic CC method due to the *quadratic effect*, that is, the probability of failed validation increases as the square of txn size. Several two-phase processing methods are described and shown to outperform restart-oriented locking methods in high-contention environments provided adequate hardware resources are available. This tutorial reviews CC methods based on standard locking, restart-oriented locking methods, two-phase processing methods including optimistic CC, and hybrid methods (combining optimistic CC and locking) in centralized systems. Its main goals are as follows: (i) succinctly specify CC methods of interest; (ii) describe models for performance evaluation of CC methods, including new models that alleviate some of the shortcomings of models used in earlier studies; (iii) compare the performance of CC methods; (iv) list insights gained from analytic and simulation studies; (v) review methods to relieve the level of lock contention: special methods for indices and aggregate data; modified txn structures; and relaxed levels of consistency for queries; (vi) survey performance evaluation studies of CC methods; (vii) illustrate the applicability of basic analytic methods to evaluating the performance of CC methods; and (viii) suggest areas of further investigation.

Categories and Subject Descriptors: D.4.8 [**Operating Systems**]: Performance—*Modeling and prediction; stochastic analysis; simulation*. H.2.4 [**Database Management**] Systems—*transaction processing*

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Adaptive methods, concurrency control, deadlocks, data contention, flow diagrams, load control, Markov chains, optimistic concurrency control, queueing network models, restart-oriented locking methods, serializability, thrashing, two-phase locking, two-phase processing, wait depth limited methods

CONTENTS

INTRODUCTION

The requirement for concurrency control (CC)[1] arose two decades ago to ensure correctness when a shared database is updated by multiple transactions (txn)s concurrently [Gray and Reuter 1992]. Txn concurrency or multiprogramming is required to take advantage of multiple processors and CPU-I/O overlap to attain high txn throughputs.

The universally accepted correctness criterion for processing txns against a database is *serializability*; that is, the interleaved execution of a set of concurrent txns is tantamount to a serial execution [Date 1983; Bernstein et al. 1987; Gray and Reuter 1992]. This correctness criterion is not acceptable for some applications (e.g., stock trading bids may have a FCFS (first-come, first-served) processing requirement [Peinl et al. 1988]. Such specialized systems are not considered in this tutorial. Standard locking (i.e., strict two-phase locking (2PL) with on-demand lock requests and the general waiting method on lock conflict) is almost exclusively used by current database management systems (DBMS)s [Date 1983; Bernstein et al. 1987; Gray and Reuter 1992]. Strict 2PL requires locks to be released only when the txn is committed or aborted, since this prevents cascading aborts [Bernstein et al. 1987; Gray and Reuter 1992]. Cascading aborts occur when a txn accesses an object modified by an uncommitted txn, which aborts after releasing the lock. Standard locking with some deviations is used in most commercial DBMSs [Date 1983; Gray and Reuter 1992].

We are mainly concerned with CC methods for centralized high-perfor-

[1] Abbreviations are summarized in the Glossary.

mance txn processing systems. Due to the high level of txn concurrency (required for attaining higher txn throughputs) and increasing txn complexity, both of which result in an increase in the probability of lock conflict, the performance of such systems may be limited by lock contention. In the case of standard locking this is in the form of txn blocking due to lock conflicts, which may lead to a thrashing behavior in which the majority of txns in the system are blocked. Various techniques to reduce the level of lock contention are introduced in this article, but our emphasis is on restart-oriented locking methods and two-phase processing methods [Franaszek et al. 1992].

Restart-oriented locking methods allow some txns encountering lock conflicts to be blocked, but reduce the level of lock contention by aborting other txns encountering or causing lock conflicts. Aborted txns are restarted by the system at a later time without user intervention. Restart-oriented locking methods that limit the level of txn blocking are called wait-depth-limited methods. The wait-depth-limited (WDL) method belongs to this category, since it takes into account txn progress in deciding which txn among a set of conflicting txns to abort [Franaszek et al. 1992]. Two-phase processing methods execute txns in two (or multiple) phases taking advantage of the fact that txn re-execution may not require disk I/O provided an adequately sized database buffer is available and access invariance prevails [Franaszek et al. 1990; Franaszek et al. 1992].

Two other widely discussed methods are optimistic CC (OCC) [Kung and Robinson 1981] and time-stamp ordering (TSO) methods [Ceri and Pelagatti 1984; Cellary et al. 1988], but the latter are more appropriate in distributed databases and are known to have a poor performance otherwise [Ryu and Thomasian 1986]. In fact, OCC methods are viable execution modes for two-phase txn processing, as in the case of utilizing the optimistic die and kill methods in the first and second phases, respectively [Franaszek et al. 1992].

A brief review of analytical modeling of a typical computer system model is provided to determine baseline txn performance due to hardware resource contention, that is, ignoring the effect of data contention. We also outline the performance analyses of several CC methods to illustrate the applicability of analytic techniques. Insights gained from analytic solutions are given throughout the article. The analysis of standard locking is expected to be of interest to all readers, but otherwise the analyses of restart-oriented locking methods and OCC methods and surveys of analytic studies can be skipped without loss in continuity.

Readers are expected to be familiar with basic DBMS and CC concepts. There are texts dealing with CC in databases and their performance [Bernstein et al. 1987; Tay 1987; Cellary et al. 1988; Kumar 1995; Thomasian 1996a], which are complemented by this article. This tutorial/survey should be of value to researchers interested in the performance of database systems, especially CC methods, designers of high-performance txn processing systems, and professionals concerned with tuning the performance of txn processing systems. There is also material for researchers interested in analyzing the performance of CC methods.

This article is organized as follows. In Section 1 we describe the txn execution model, the database access model, and the computer system model. In Section 2 we analyze the performance of the standard locking method, discuss its thrashing behavior, and describe load-control methods to prevent thrashing. Analyses of locking methods are next reviewed. We then introduce a more realistic locking model, which is followed by a discussion of some of the shortcomings of locking models. Finally, we describe some recently proposed methods to reduce the level of lock contention.

In Section 3 we provide a succinct description of restart-oriented locking methods, including wait-depth-limited methods and the WDL method. The performance of these and several other methods is compared through simulation and we outline the analysis of the symmetric running priority method. In Section 4 we describe OCC methods and various validation options. A specification of mechanisms utilized by two-phase processing methods is followed by the description of representative two-phase processing methods. Simulation results to evaluate the performance of two-phase processing methods against each other and with other methods are reported. Finally, we discuss analytic solutions for OCC methods in Section 4.4.

In Section 5 we summarize the major points of the earlier sections of the article. We next discuss difficulties in developing analytic models for predicting performance degradation due to locking in operational txn processing systems. Finally, several areas requiring further investigation and some new areas where CC is an issue are listed.

The notation required for analyzing the performance of CC methods is introduced in Section 1 and summarized in the Appendix.

1. MODELING AND ANALYSIS OF TRANSACTION PROCESSING SYSTEMS

We describe models for evaluating the performance of txn processing systems as affected by hardware and data resource contention. Hardware resource contention has the primary effect on performance, whereas data contention has a secondary effect. Relatively abstract models, both from the viewpoint of data and hardware resource contention, are used in performance analyses and simulation studies of CC methods. These models are not detailed enough for predicting the performance of "real" txn processing systems, but are deemed adequate for the performance comparison of CC methods.

Sections 1.1 through 1.3 describe the txn, database access, and computer system models. The reader is referred to Chapter 2 in Thomasian [1996a] for a much more detailed discussion of analytic and simulation methods to evaluate the performance of CC methods.

1.1 Transaction Model

Transaction Execution Steps. Single-level or flat txns, as opposed to multilevel txns [Gray and Reuter 1992], and relatively short update txns, rather than batch update txns or read-only queries, are of main interest in this study (other txn structures are discussed in Elmagarmid [1992] and Ramamithram and Chrisanthis [1996]). Techniques to cope with the lock contention between long-running read-only queries and short update txns are discussed in Section 2.5. A txn accessing k database objects consists of $k + 1$ execution steps, numbered 0–k. Each step involves CPU processing and disk accesses. The last k steps begin with a database call, which leads to an access to a single database object, after an appropriate lock on the object has been acquired. The completion of the last step leads to txn *commit* (i.e., the writing of log records onto stable storage) after which the locks held by the txn are released [Date 1983; Bernstein et al. 1987; Gray and Reuter 1992]. Logging of commit data onto disk adds to the txn response time and the holding time of locks, but logging time can be reduced by providing a nonvolatile RAM (random access memory) for this purpose [Franaszek et al. 1992]. The number of objects accessed by a txn is referred to as txn size. A system may process multiple txn classes, where txn class k is denoted by C_k. Txn class is determined by its size in this work, although other txn characteristics may be used for this purpose.

Transaction arrival process. Txns originate from a finite number of sources (say S), which may correspond to bank tellers generating txns after a think time with a mean Z; that is, the arrival rate with M txns already at the system is $\Lambda(M) = (S - M)/Z$. We consider *one-shot* txns, such that the user is not involved after the txn is submitted; that is, there are no intervening user think times [Agrawal et al. 1987a]. Quasirandom arrivals with exponentially distributed think times lead to analytically tractable models [Kleinrock 1975]. A truncated exponential distribution is adopted in txn processing benchmarks to avoid very long think times [Gray 1993].

A system with a sufficiently large S is considered to be *open* with a txn arrival rate independent of the number of txns at the system. The txn arrival process in an open system is usually assumed to be Poisson with an arrival rate λ [Kleinrock 1975]. The *mean response time characteristic* (i.e., the mean txn response time $R(\lambda)$ versus λ) can be used for the performance comparison of CC methods in this case.

A system with $S = M$ sources is considered to be *closed* when $Z = 0$. A completed txn is then immediately replaced by a new txn so that the degree of txn concurrency in the system is always M. The seemingly unrealistic closed model is quite useful in comparing the performance of CC methods because: (i) the solution of a closed model can be used as a submodel in performance evaluation of a system with external arrivals [Lazowska et al. 1984]; (ii) it is easier to estimate the peak performance of CC methods by simulating a closed rather than an open model, which is due to the variability of the number of txns in an open model. The performance of a closed system with M txns can be specified by its throughput $T(M)$, whereas $T(M)$, $M \geq 1$ is referred to as its *effective throughput characteristic* (ETC). It follows from Little's result (that the mean number of requests in a system is equal to the product of the arrival rate of requests and the mean time they spend in the system [Kleinrock 1975; Lazowska et al. 1984]) that the mean residence time of txns in the system is $R(M) = M/T(M)$.

A *frequency-based model* specifies the fraction f_k of txns in C_k processed by the system. In an open system the arrival rate of txns in C_k is $\lambda_k = \lambda f_k$, $1 \leq k \leq K$, where K is the largest txn size. Note that $\sum_{k=1}^{K} f_k = 1$ with $f_k > 0$ for txn sizes being processed by the system and $f_k = 0$ otherwise. The mean number of txns in C_k is $\bar{M}_k = \lambda_k R_k$ according to Little's result, where R_k is the mean response time of txns in C_k.

In a closed system we may specify the degree of txn concurrency M_k of txns in C_k. A frequency-based model is also applicable in this case, with a completed txn being immediately replaced by a new txn in C_k with probability f_k. The throughput for txn completions in C_k is a fraction of f_k of $T(M)$; that is, $T_k(M) = f_k T(M)$, $1 \leq k \leq K$. The performance comparison of CC methods with a frequency-based closed system simply requires the comparison of their ETCs [Ryu and Thomasian 1987; Thomasian 1993], and an open model requires the comparison of the mean-response-time characteristics, which are more difficult to obtain (see the hierarchical solution method in Section 1.3).

1.2 Database Access Model

The *granule* is the unit of data at which CC is applied to the database. A granule may be associated with multiple database objects; for example, a page-level lock applies to all the records in a page [Gray and Reuter 1992]. The effect of granularity of locking on performance has been considered in several studies (see, e.g., Tay [1987]). A finer level of granularity of locking incurs more overhead, but reduces the level of lock contention. For example, the frequency of lock conflicts in processing short txns is lowered with record- rather than page-level locking, whereas a query accessing a relational table should preferably use table-level locking since with record- and even page-level locking it would incur a tremendous overhead [Gray and Reuter 1992; Thomasian 1996a]. Techniques to resolve the lock contention between txns and read-only queries are discussed in Section 2.6.

We consider a database with D objects that are locked in exclusive and shared mode. This simple locking model can be justified by the fact that we are mainly concerned with short txns.

Most studies assume that database objects are accessed with uniform probabilities. Nonuniformity of database access is captured by the hot-spot model and the $b - c$ rule [Tay 1987]; that is, a fraction b (resp., $1 - b$) of txn accesses are to a fraction c (resp., $1 - c$) of the database.

A homogeneous database access model is postulated in most studies such that all txn classes follow the same lock-request pattern (see Section 2.1). A more realistic database access model for txn processing with multiple txn classes and multiple database regions is based on the heterogeneous database access model, which is elaborated in Section 2.5.

Sequential database access has been considered in some early simulation studies of static locking (see Chapter 4 in Tay [1987]) and is analyzed in Ryu and Thomasian [1988]. Database access patterns or granule placements are reviewed in Langer and Shum [1982].

Performance analyses of CC methods can be classified according to whether the objects accessed and locks obtained by a txn are distinct or not, which are referred to as without- and with-replacement options [Langer and Shum 1982]. Analyses in the former case lead to expressions with binomial coefficients, which can be approximated accurately by simpler expressions for the with-replacement option (see the expression for probability of data conflict in Section 4.4). When the number of lock requests by a txn (say k) is much smaller than D, then even if locks are replaced, the chances that a txn will request the same lock are very small. This is why expressions for the probability of data conflict for with- and without-replacement options yield indistinguishable results. Expressions for the with-replacement option are preferable, since they are easier to evaluate [Thomasian 1996a].

1.3 Computer System Model

A queueing network model (QNM) determines the execution time of txns in the computer system by taking into account its processing time at the various devices of the computer system, as well as queueing effects. The processing time at the CPU is determined by txn pathlength and CPU speed. Disk processing time is determined by the disk access time and the number of disk I/Os, which is affected by the miss ratio of the database buffer [Gray and Reuter 1992]. The buffer hit ratio of a workload is a function of its buffer space allocation and the buffer management policy, including the buffer replacement policy, for example, least recently used, LRU [Gray and Reuter 1992].

The infinite-resource model is useful in comparing the performance limits of CC methods [Franaszek and Robinson 1985; Ryu and Thomasian 1987; Tay 1987; Thomasian 1993]. According to this model, each txn has its own virtual processor and its execution time is independent of the number of txns being executed concurrently. The mean residence time of txns in the system is then $r(M) = c$ and the system throughput is $t(M) = M/c$, $M \geq 1$.

A finite-resource model allows the level of hardware resource contention to be varied to determine its effect on overall performance [Thomasian and Ryu 1986; Agrawal et al. 1987; Franaszek et al. 1992; Thomasian 1993]. This is especially important when comparing the performance of restart-oriented CC methods with blocking-oriented CC methods (such as standard locking), where the former introduce wasted processing (processing not leading to the successful completion of a txn), whereas the latter may result in system underutilization due to blocked txns.

Figure 1 shows a set of user terminals accessing a computer system with the

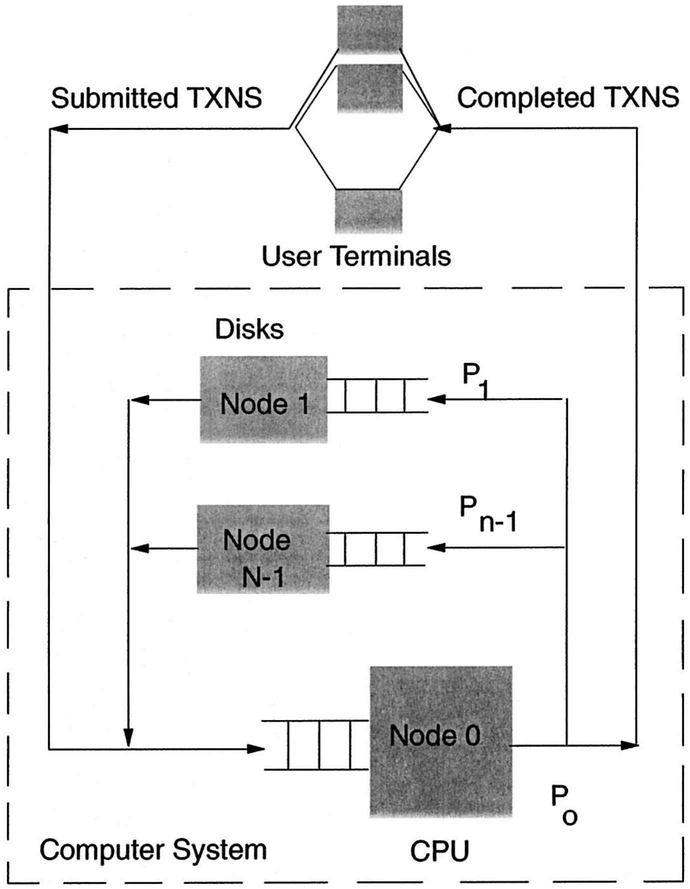

Figure 1. Central server queueing model.

central server QNM, where the central server is the CPU and the peripheral servers are the disks. The probability that a txn completes after the current CPU burst and a response is sent to a user terminal is p_0. The number of visits to the CPU follows the geometric distribution $P_k = p_0(1 - p_0)^{k-1}$, $k \geq 1$, with the mean equal to $1/p_0$. After completing CPU processing, a txn accesses the nth disk with probability p_n such that the mean number of visits to disk n is p_n/p_0. This simple model for txn transitions does not lend itself to modeling fixed-size txns. This is accomplished by making txn transitions at the completion of CPU processing dependent on the state of the txn [Irani and Lin 1979; Thomasian 1982].

QNMs can be analyzed to determine the response time of txns as affected by hardware resource contention [Kleinrock 1975; Lazowska et al. 1984]. Product-form QNMs constitute a small subset of QNMs, since several strict requirements need to be met for a QNM to be product-form (e.g., the service time distribution at nodes with the FCFS discipline is exponential [Lazowska et al. 1984]). Performance measures such as throughput, device utilizations, and mean queue lengths can be computed efficiently for a product-form QNM using the mean service demands of txns at its nodes. The mean service demand of a txn at a node is the product of the mean number of visits the txn makes to the node and the mean service time per visit. For a given txn arrival rate the mean residence time at the nodes of an open QNM can be computed separately, as if the arrival process to each node were Poisson [Kleinrock 1975]. In the case of a closed QNM, the convolution or mean-value analysis algorithm [Lazowska et al. 1984] can be used to obtain the system throughput characteristic (STC) $t(m)$, $1 \leq m \leq M_{\max}$, where $M_{\max}$ is the maximum degree of txn concurrency.

The device with the highest utilization in a QNM is referred to as the *bottleneck* device. It is the first to be saturated as the arrival rate (resp., the number of txns) is increased in an open (resp., closed) system. The maximum throughput is given by $t_{\max} = 1/X'_{\text{bottleneck}}$, where $X'_{\text{bottleneck}} = \max(X_1/m_1, \ldots, X_N/m_N)$, with X_n being the mean service demand and m_n the number of servers at node n [Kleinrock 1975; Lazowska et al. 1984].

A realistic computer system model for txn processing cannot be adequately represented by a product-form QNM, for example, nonexponential service times at disks with a FCFS discipline, CPU priorities, and txn blocking due to lock conflicts [Franaszek et al. 1992].

Hierarchical solution method. The hierarchical solution method is useful in evaluating the performance of txn processing systems with external arrivals, which cannot be solved directly as open models although the underlying QNM is product-form, for example, due to constraints on the degree of txn concurrency or lock conflicts in static locking [Thomasian and Ryu 1983; Thomasian 1985] (see Section 2.3). The theoretical justification for this solution method is the decomposition or aggregation principle in QNMs [Lazowska et al. 1984]. For example, the computer subsystem in Figure 1 can be replaced by an aggregate or flow-equivalent service center [Lazowska et al. 1984]. In the case of a single job-type (txn class) the aggregate server is state-dependent with a completion rate given by the STC ($t(M)$, $M \geq 1$).

The processing of txns with Poisson arrivals can be represented by a birth–death model, with states corresponding to the number of txns at the system (M) [Kleinrock 1975]. The birth rate at all states is λ and the death rate at state M is $t(M)$. The birth–death model has a solution provided $t_{\max} > \lambda$, where $t_{\max}$ is the maximum attainable throughput. A system with quasirandom arrivals can be represented with a birth–death model with $S + 1$ states and arrival rates $\Lambda(M) = (S - M)/Z$, $0 \leq M \leq S$.

Multiple txn classes that have different processing requirements can be specified as different job types in QNM terminology. The flow-equivalent service center in the case of two job types is specifiable as $t_1(M_1, M_2)$, $t_2(M_1, M_2)$, $M_1 \geq 0$, $M_2 \geq 0$. Such flow-equivalent servers have been utilized in the analyses in Potier and LeBlanc [1983], Thomasian [1985], and Thomasian [1993a].

In a single-txn-class system with data contention, the ETC ($T(M)$, $M \geq 1$) rather than the STC ($t(M)$, $M \geq 1$) should be used in the analysis.

Performance degradation due to CC Methods. This degradation is due to txn blocking, restarts, or both. In the following discussion any overheads associated with txn blocking and aborts are ignored. The following cases are possible.

(1) *CC methods with no (or negligible) wasted processing.* Standard locking with dynamic lock requests meets this criterion, since txn aborts to resolve deadlocks are rare and txn blocking due to lock conflicts incurs a negligible overhead. An analysis may yield the mean number of active (versus blocked) txns $\bar{M}_a$ in a system with M txns [Tay et al. 1985]. Provided that the STC is not highly discontinuous, the effective throughput is given by $T(M) \simeq t(\bar{M}_a)$. Interpolation can be used when $\bar{M}_a$ is not an integer.

(2) *CC methods with restarts but no blocking.* OCC methods fall into this category. The system efficiency is the fraction of useful processing in the system and can be expressed as $T(M)/t(M)$ (see Section 3.3 for a more detailed discussion).

(3) *CC methods with blocking and restarts.* The running priority method and WDL belong to this category [Franaszek and Robinson 1985; Franaszek et al. 1992]. Since the wasted processing is due to active txns, the system efficiency is defined as $T(\bar{M}_a)/t(\bar{M}_a)$. The method with the highest efficiency in this case may not be the method maximizing $\bar{M}_a$ and attaining the maximum throughput with an infinite resource model (see Section 3.2).

Separation of hardware and data-resource contention. The separation of hardware and data contention is a desirable property, since it allows the STC to be computed only once. This separation is possible if the data contention overhead is insignificant or is independent of the level of lock contention. Wasted processing, as in the case of restart-oriented locking methods, allows this separation provided the aforementioned conditions are met [Tay et al. 1985; Tay 1987; Thomasian 1992, 1995a].

The overhead associated with different CC methods is comparable [Robinson 1984]. This overhead may be included in the processing associated with txn steps, but there is a dependence on the level of data contention; for example, a lock request leading to a conflict and txn blocking requires additional CPU processing due to a context switch. In static locking extra processing is required to check for the eligibility of blocked txns for activation [Potier and LeBlanc 1980; Thomasian and Ryu 1983] (see Section 2.3). Txn aborts require extra processing to undo txn's updates and restart the txn.

Lock contention depends on lock holding times, which are affected by hardware resource contention, whereas hardware-resource contention is determined by the number of active txns and the processing requirements for txn steps, which are affected by the lock-contention level. An iterative solution combining the analysis of the hardware-resource-contention model with the data-contention model is therefore required [Ryu and Thomasian 1990a].

2. STANDARD LOCKING AND ITS PERFORMANCE

In standard locking, locks are usually requested on demand, which is referred to as *dynamic locking* (DL), whereas in *Static Locking* (SL) locks for all database objects required for the execution of a txn are assumed to be known a priori and are requested before the txn begins its execution. Strict 2PL with locks released at txn completion time is considered in this study, but some variations to strict 2PL are discussed in Section 2.6.

The *general waiting* (GW) method, according to which a txn making a conflicting lock request[2] is blocked awaiting the release of the lock, is susceptible to encountering a deadlock, where two txns are blocked indefinitely awaiting the release of each other's locks. Deadlocks are resolved by aborting one of the txns in a deadlock cycle according to one of the victim selection policies [Agrawal et al. 1987b]:

(1) the txn that is the current blocker;
(2) a random blocker (i.e., one of the txns involved in the deadlock cycle);
(3) the txn with the minimum number of locks;
(4) the youngest txn; and

[2] A lock conflict occurs when the requested lock is held in an incompatible mode by another txn or there is a pending lock request in an incompatible mode for it; for example, the lock is held in shared mode and there is an exclusive lock request pending for it when another txn requests a shared lock on it. Lock compatibility tables are given in Date [1983], Bernstein et al. [1987], and Gray and Reuter [1992].

(5) the txn with the minimum amount of work.

It follows from simulation results reported in Agrawal et al. [1987b] that the deadlock resolution policy has little effect on the maximum attainable throughput of the standard locking method.

Cyclic restarts occur when restarted txns have lock conflicts with txns with which they have had lock conflicts before and this leads to repeated deadlocks. Cyclic restarts are a possibility with policies (1) and (2), whereas policy (4) guarantees that they are prevented. Policies (3) and (5) are pseudostable in that they tend to prevent cyclic restarts, but do not guarantee this. Restart waiting prevents cyclic restarts by delaying the restart of an aborted txn until the conflicting txns have left the system. This method is adopted in simulation studies reported in Ryu and Thomasian [1990a], Thomasian and Ryu [1991], Franaszek et al. [1992], and Thomasian [1993, 1997a].

Only shared and exclusive locks are considered, since we are concerned with lock contention among relatively short txns (as noted in Section 1.2). A txn holding a shared lock on an object should promote it to an exclusive lock before it can update the object. Deadlocks will arise if this action is taken by more than one txn holding a shared lock on the same object; 97% of deadlocks in a database prototype were due to such lock promotion [Date 1983]. When the updating of an object is a possibility, txns can prevent deadlocks by acquiring update locks, since they are compatible only with shared locks, not with each other and exclusive locks [Date 1983; Gray and Reuter 1992]. The analysis of DB2 traces for several workloads reveals that approximately 90% of exclusive locks are obtained directly [Singhal and Smith 1997]. This can be used as a justification for not considering lock promotion and update locks.

The simulation study for validating analytic solutions detects and resolves deadlocks immediately. Some systems carry out deadlock detection on a periodic basis [Gray and Reuter 1992], which is more appropriate for distributed databases, since extra messages are required for deadlock detection [Ceri and Pelagatti 1984]. Timeouts are suitable for deadlock resolution in distributed systems and shared-nothing architectures, but this method has the drawback that the timeout interval is difficult to determine [Jenq et al. 1989].

This section is organized as follows. In Section 2.1 we derive the probability of lock conflict, probability of deadlock, and the mean blocking time per lock conflict with respect to an active txn. In Section 2.2 we analyze the performance of DL and demonstrate its thrashing behavior. In Sections 2.3 (resp., 2.4) we survey analytic solutions for SL (resp., DL). In Section 2.5 we introduce a heterogeneous database access model. We also discuss some shortcomings of current locking models from the viewpoint of evaluating the performance of txn processing systems. Finally, in Section 2.6 variations to standard locking are described.

2.1 Lock Conflicts and Deadlocks

We first analyze a system with fixed txn sizes and then extend the analysis to variable txn sizes. A closed system with M txns is considered in both cases.

Analysis of dynamic locking with fixed-size transactions with identical per-step processing times. The mean txn response time is the sum of its mean execution time and the mean blocking times when it encounters a lock conflict. The increase in txn response time due to restarts to resolve deadlocks is ignored, since the frequency of deadlocks tends to be small [Thomasian and Ryu 1991] (also see discussion in Section 2.2). The mean blocking time per step is $u = P_c W$, where P_c is the probability of lock conflict (per lock request) and W is the mean waiting time (per lock conflict). Noting that there is no blocking in the first step, the mean txn response time is $R(M) = (k + 1)s(\bar{M}_a) + kP_cW$ and the effective txn throughput is $T(M) = M/R(M)$.

The mean number of locks ($\bar{L}$) held per txn is the ratio of the mean time-space of locks held by txns and the mean txn response time. In a system with no lock contention $\bar{L} = k/2$, and $\bar{L} \simeq k/2$ is a good approximation for a system with lock contention [Ryu and Thomasian 1990a]. When all lock requests are exclusive the probability of lock conflict (P_c) for the ith lock request is

$$P_c = \frac{\text{Mean number of locks held by other txns}}{\text{Number of database locks not held by the txn}}$$

$$= \frac{\bar{N} - i}{D - i} \simeq \frac{(M-1)\bar{L}}{D} \simeq \frac{(M-1)k}{2D}, \tag{2.1}$$

where $\bar{N} \simeq (M - 1)\bar{L}$ is the mean number of locks held by the other $M - 1$ txns in the system, where each txn contributes an average number of locks. We have taken advantage of the fact that txn size $k(\geq i)$ tends to be much smaller than the database size (D), so that the dependence of P_c on the txn step has been ignored [Tay 1987; Ryu and Thomasian 1990a; Thomasian 1993]. The probability that a txn encounters a lock conflict is

$$P_w = 1 - (1 - P_c)^k \simeq kP_c \simeq \frac{(M-1)k^2}{2D}. \tag{2.2}$$

The approximation is justified by the fact that P_c tends to be small (e.g., less than 0.001). The probability that two txns are involved in a two-way deadlock is [Gray and Reuter 1992]

$$P_D(2) = \Pr[T_1 \rightarrow T_2]\Pr[T_2 \rightarrow T_1] \cdot (\text{number of candidates for } T_2)$$
$$= \frac{P_w{}^2}{M-1} \simeq \frac{(M-1)k^4}{4D^2}. \tag{2.3}$$

Similar expressions apply to multiway deadlocks, but since P_w is very small, $P_D(i)$ for $i > 2$ is negligibly small so that $P_D = \sum_{i \geq 2} P_D(i) \simeq P_D(2)$. A more accurate expression for $P_D(2)$ is obtained in the following.

The fraction of time txns are blocked in the system (β) is the ratio of txn blocking time and its mean response time, which is equal to the fraction of blocked txns in the system:

$$\beta = \frac{kP_cW}{R(M)} = \frac{\bar{M}_b}{M}. \tag{2.4}$$

The equality follows from Little's result by multiplying the numerator and denominator of the first fraction by $T(M)$.

In a system with a low lock-contention level, most lock conflicts are with active txns. The mean waiting time W_1 can be obtained by noting that the probability of lock conflict increases with the number of locks (j) that the (active) txn holds [Thomasian and Ryu 1991]:

$$W_1 = \sum_{j=1}^{k} \frac{2j}{k(k+1)} [(k-j)(s(\bar{M}_a) + u) + s'(\bar{M}_a)] = \frac{k-1}{3}[s(\bar{M}_a) + u] + s'(\bar{M}_a), \tag{2.5}$$

where $k(k + 1)/2$ is a normalization constant and $s(\bar{M}_a)$ is the mean duration of txn steps, which is determined by analyzing or simulating the corresponding QNM with $\bar{M}_a$ active txns. The term in brackets is the mean remaining processing time after a lock conflict occurs at the jth step of txn execution and

$s'(\bar{M}_a)$ is the mean residual processing time of the step in which the lock conflict occurred [Kleinrock 1975].[3] A similar expression is derived in Tay et al. [1985] and Tay [1987]. The fraction of time that a txn is blocked by an active txn is $A = W_1/R(M)$. In the case of fixed-size txns, it follows from Eq. (2.5) that $A \simeq 1/3$.

Eq. (2.3) does not take into account the fact that T_1 should be in the blocked state for a deadlock to occur. In the case of two-way deadlocks, the fraction of time the other txn is blocked is $A \simeq 1/3$, which leads to $P'_D(2) \simeq P_D(2)/3$ [Thomasian and Ryu 1991], where $P_D(2)$ is given by Eq. (2.3). The analysis in Massey [1986] yields $2P'_D(2)$, and that in Tay et al. [1985] and Tay [1987] yields $4/3P'_D(2)$.

Analysis of dynamic locking with variable-size transactions. We distinguish txn classes based on their size, as determined by the number of requested locks. As stated in Section 1.1, the fraction of txns requesting k locks is denoted by f_k, $1 \le k \le K$ with $\sum_{k=1}^{K} f_k = 1$, where K denotes the largest txn size. The ith moment of the number of requested locks is $K_i = \sum_{k=1}^{K} k^i f_k$.

The mean response time for txns in C_k is $R_k(M) = (k + 1)s(\bar{M}_a) + kP_cW$, from which the mean response time over all txn classes is given as

$$R(M) = \sum_{k=1}^{K} R_k(M) f_k = r(\bar{M}_a) + K_1 P_c W, \tag{2.6}$$

where $r(\bar{M}_a) = (K_1 + 1)s(\bar{M}_a)$. It follows from Eq. (2.4) that $\beta = K_1P_cW/R(M)$ in the case of variable size txns, hence

$$R(M) = r(\bar{M}_a)/(1 - \beta), \tag{2.7}$$

which also applies to fixed-size txns. The effective throughput can be obtained from the STC $T(M) \simeq t(\bar{M}_a)$, since the wasted processing due to deadlocks is negligibly small.

P_c is determined by the mean number of locks held by a txn, which is affected by the distribution of txn size. It follows from $\bar{M}_k/M = f_kR_k(M)/R(M)$ that $\bar{M}_k \simeq Mkf_k/K_1$.[4] The mean number of locks held per txn is then

$$\bar{L} = \frac{1}{M}\sum_{k=1}^{K} \bar{L}_k\bar{M}_k \simeq \frac{1}{M}\sum_{k=1}^{K} \frac{k}{2}\bar{M}_k$$

$$\simeq \frac{1}{2K_1}\sum_{k=1}^{K} k^2 f_k = \frac{K_2}{2K_1}. \tag{2.8}$$

In the case of the geometric distribution $f_k = q(1 - q)^{k-1}$, $k \ge 1$. Since $K_1 = 1/q$ it follows that setting $q = 1/K$ yields $K_1 = K$ and $\bar{L} \simeq K$, as opposed to $\bar{L} \simeq K/2$ for fixed-size txns; that is, P_c for this distribution is twice that for fixed txn sizes [Thomasian and Ryu 1991].

The probability of deadlock per txn is affected by the third moment of txn size [Thomasian and Ryu 1991]; for example, this probability is an order of magnitude higher for geometrically distributed txn sizes with the same mean as fixed-size txns.

$A = W_1/R(M)$ can be expressed as follows [Thomasian and Ryu 1991; Thomasian 1993].

$$A = \frac{W_1}{R(M)} \simeq \frac{K_3 - K_1}{3K_1(K_2 + K_1)}, \tag{2.9}$$

which ignores the residual delay in the step in which the lock conflict occurred.

Validation against simulation has shown the accuracy of the analysis in estimating the preceding variables for

[3] $s'(\bar{M}_a)$ equals $s(\bar{M}_a)/2$ when the processing time is fixed, $2s(\bar{M}_a)/3$ when it is uniformly distributed [Tay 1987], and $s(\bar{M}_a)$ when it is exponentially distributed, which is due to the memoryless property of this distribution [Kleinrock 1975].

[4] The approximation is due to the fact that txn response times in the numerator and denominator are proportional to $k + 1$ and $K_1 + 1$, rather than k and K_1, respectively.

fixed- and variable-size txns [Thomasian and Ryu 1991].

2.2 A Performance Analysis of Dynamic Locking

We analyze the performance of DL first with identical and then different per-step processing times. Factors contributing to thrashing are then discussed.

The effect of deadlocks on system performance is ignored in this analysis since deadlocks tend to be rare (as shown in Section 2.1) and taking this effect into account requires a rather complicated analysis, as in Ryu and Thomasian [1990a]. Ignoring the wasted processing due to txn restarts to resolve deadlocks is justifiable by the fact that it constitutes a small fraction of the total processing cost [Thomasian 1993]. According to Eq. (2.3), the probability of deadlock increases with the fourth power of txn size, which is a weakness of the analysis for longer txns; that is, the mean response time for these txns is underestimated when the lock-contention level is high.

Analysis with identical per-step processing times. Txn blocking can be specified by a *waits-for graph* (WFG), which is a directed graph with nodes representing txns and edges the waits-for relationship. In a system with only exclusive lock requests, the WFG is a directed tree, with each blocked txn pointing to an active txn or another blocked txn holding the requested lock (see, e.g., Tay et al. [1985] and Tay [1987]). The system state can be represented by a forest of trees, where the nodes with out-degree zero represent active txns. Active txns are considered to be at the roots of the trees and are designated to be at level zero, txns blocked by active txns are at level one, txns blocked by level-one-blocked txns are at level two, and so on. The number of levels by which blocked txns are removed from active txns or their level in the tree is referred to as their *wait depth*.

If shared locks in addition to exclusive locks are considered, the WFG is no longer a directed tree but rather a directed acyclic graph, since a txn might be blocked by multiple active txns holding the same shared lock. The evolution of the WFG depends on the lock-scheduling policy. FCFS scheduling assures fairness and a threshold scheduler for locks will increase the degree of concurrency in processing shared-lock requests [Thomasian and Nicola 1993].[5] This results in an increase of the maximum throughput at which such requests can be processed by the system [Thomasian and Nicola 1993]. NonFCFS scheduling of lock requests is not considered further at this point, but lock priorities are considered in Section 4.

Simulation studies show that the wait depth tends to be limited to a few levels for reasonable simulation parameters (e.g., txn sizes much smaller than database size) so we first consider the case when a txn is blocked either by an active txn or a txn that is blocked (by another active txn). The probability of being blocked by an active (resp., blocked) txn is $1 - \beta$ (resp., β) and the mean blocking time is W_1 (resp., $W_2 = 1.5W_1$).[6] More generally, the probability that the effective level of blocking of a

[5] The threshold scheduler processes shared-lock requests without regard for enqueued exclusive-lock requests, until the number of exclusive-lock requests exceeds a prespecified threshold. At this point, in-progress shared lock requests are processed to completion and further shared lock requests are enqueued. All exclusive-lock requests are processed next, before the system reverts to processing shared-lock requests. The cycle repeats. This scheme requires timeouts to avoid long delays in processing lock requests of either type.

[6] This is informally justified as follows. Consider a txn T_2 that has a lock conflict with a blocked txn T_1; for example, $T_2 \rightarrow T_1 \rightarrow T_0$. T_2 encounters its lock conflict with T_1 at a time distributed uniformly over the time T_1 was blocked by T_0, hence the extra delay is $0.5W_1$. T_2 may have been initially blocked by T_1, which became blocked at a later time with T_0, resulting in the WFG $T_2 \rightarrow T_1 \rightarrow T_0$ as before. It can be argued that T_2 had completed one half of its waiting time when T_1 was blocked.

txn (as opposed to its initial level of blocking) is i, is approximated by $P_b(i) = \beta^{i-1}$, $i > 1$ and $P_b(1) = 1 - \beta - \beta^2 - \beta^3 \ldots$. The mean waiting time at level $i > 1$ is approximated by $W_i = (i - 0.5)W_1$. The mean overall waiting time (W) is a weighted sum of delays incurred by txns blocked at different levels:

$$W = \sum_{i \geq 1} P_b(i) W_i$$

$$= W_1 \left[1 - \sum_{i \geq 1} \beta^i + \sum_{i > 1} (i - 0.5)\beta^{i-1} \right]. \quad (2.10)$$

We define $n_c = K_1 P_c$ as the mean number of lock conflicts per txn. Multiplying both sides by $n_c/R(M)$ and defining $\alpha = n_c A$ (with $A = W_1/R(M)$) and since $\beta = n_c W/R(M)$, we have

$$\beta = \alpha(1 + 0.5\beta + 1.5\beta^2 + 2.5\beta^3 + \ldots). \quad (2.11)$$

At low lock-contention levels $\beta \simeq \alpha$, such that $\bar{M}_a \simeq M(1 - \alpha)$. The approximation $R(M) \simeq r(M)/(1 - \alpha)$ for closed systems or $R(\lambda) \simeq r(\lambda)/(1 - \alpha)$ for open systems has been used in numerous performance studies of txn processing systems. A better approximation for β can be obtained by substituting β with α on the RHS of Eq. (2.10) [Thomasian and Ryu 1991].

We can obtain a closed-form expression for the series in Eq. (2.11) by assuming that it is infinite, since it converges for $\beta < 1$, leading to

$$\beta^3 - (1.5\alpha + 2)\beta^2 + (1.5\alpha + 1)\beta - \alpha = 0. \quad (2.12)$$

It is stated in Tay [1990] that "As yet, there are no proposed measures for the resource requirements of a given concurrency control algorithm." Note that α, which is the product of the mean number of lock conflicts per txn and the mean waiting time per lock conflict (with respect to active txns) normalized by mean txn response time, is a single metric that determines the level of lock contention for standard locking. Two different systems will have the same lock contention level as long as they have the same α.

The cubic equation has an algebraic solution that yields $0 \leq \beta_1 < \beta_2 \leq 1$ and $\beta_3 \geq 1$ for $\alpha \leq \alpha^* = 0.226$ and a single root $\beta_3 > 1$ for $\alpha > \alpha^*$. α^* can be used as an indicator of whether the system is operating in the thrashing region. The smallest root β_1 for $\alpha \leq \alpha^*$ determines system performance.

Two cases are possible when the STC ($t(m)$, $m \geq 1$) is a nondecreasing function, before it attains a fixed value due to the saturation of the bottleneck resource (see Section 1.3): the system is hardware-resource-bound (e.g., the processor saturates while the level of lock contention in the system is low), or the system is data-contention-bound and thrashes before the hardware resource bound is attained. In the latter case, the effective throughput $T(M)$ increases with M and the peak throughput is attained when $\bar{M}_a = M(1 - \beta)$ is at its maximum. Plotting $\bar{M}_a$ versus β for several txn size distributions, it is observed that $\bar{M}_a$ is maximized at $\beta \simeq 0.3$ [Thomasian 1993]. This is also verified from $d\bar{M}_a/d\alpha = 0$, which yields $\hat{\alpha} = 0.2135$ with a corresponding $\hat{\beta} \simeq 0.3$. The ETC ($T(M)$), $M \geq 1$) increases with M, reaches a peak at $\hat{\alpha}$ corresponding to $\hat{M}$, at which point $\bar{M}_a$ and hence $T(\hat{M}) \simeq t(0.7\hat{M})$ (assuming infinite hardware resources) reach their maximum.

If we subject the system to quasirandom arrivals (see Section 1.1), then the intersection point of $\Lambda(M) = (S - M)/Z$, $1 \leq M \leq S$ with the ETC $T(M)$, $1 \leq M \leq S$, corresponds to the equilibrium point of the system, where the txn arrival rate is equal to the completion rate. There may be one intersection point in the stable or thrashing region, or two points, in which case the system is bistable [Tay 1987; Thomasian 1993].

The value of $\hat{\alpha}$ can be used for *load control*, that is, preventing thrashing by limiting the number of txns activated in the system. Since the value of α cannot be estimated directly, the critical value of the mean number of lock conflicts per txn ($\hat{n}_c = \hat{\alpha}/A$) can be used for this purpose, provided A is known. In the case of fixed-size txns $A \simeq 1/3$ and $\hat{n}_c \simeq 0.64$, as opposed to $\hat{n}_c \simeq 0.75$ observed from simulation results [Tay et al. 1985; Tay 1987]. In the case of the geometric distribution, $A \simeq 1$ according to Eq. (2.9) and $\hat{n}_c \simeq 0.21$. In general, the value of A computed from this equation using the first three moments of txn size distribution (K_i, $1 \le i \le 3$) can be used to estimate $\hat{n}_c$. Also the critical value of the probability of lock conflict per lock request $\hat{P}_c = \hat{n}_c/K_1$ is an easy-to-use metric for detecting thrashing.

Note that the variability of txn size has a major effect on system performance; for example, in a system with infinite resources the peak txn throughput with a (truncated) geometric distribution is a factor of three smaller than the peak throughput with fixed-size txns [Thomasian 1993]. In fact the value of α for fixed-size txns is one sixth of the α for geometrically distributed txns with the same mean size.

This analysis is shown to be quite accurate through validation, except at the highest contention level at the extreme cases (i) small M and large k (large txns); and (ii) large M and small k. The problem in the first case is that (i) the analysis with mean values is not applicable to a system with a few txns; (ii) the analysis does not take into account the fact that a single txn may block multiple txns; that is, the wait depth is over-estimated by the analysis. Not surprisingly, the analysis underestimates the performance in this case. A detailed analysis to estimate W is undertaken in Tay et al. [1985] and Tay [1987]. The accuracy of global performance measures estimated by this analysis is comparable to the analysis in this section, which is based on Thomasian [1993].

Analysis with different per-step processing times. The mean processing time of the ith step of a txn in C_k is denoted by $s_i^k(\underline{\bar{M}}_a)$, $0 \le i \le k$, $1 \le k \le K$, where $\underline{\bar{M}}_a$ is the vector of the mean number of txns in different classes and steps, which determines the duration of txn steps. The mean txn response time can be obtained from Eq. (2.6), once P_c, W, and $\underline{\bar{M}}_a$ are determined.

The probability of lock conflict with a blocked txn with identical per-step processing times can be approximated by β, since active and blocked txns approximately hold the same number of locks [Ryu and Thomasian 1990a]. When txn steps have different processing times this probability is approximated by $\rho \simeq \bar{L}_b/\bar{L}$, where $\bar{L} = \bar{L}_a + \bar{L}_b$ with $\bar{L}_a$ and $\bar{L}_b$ denoting the mean number of locks held by active and blocked txns, respectively, which can be computed as

$$\bar{L}_a = \sum_{k=1}^{K} f_k \sum_{i=1}^{k} \frac{i s_k^i(\underline{\bar{M}}_a)}{R(M)},$$

$$\bar{L}_b = \sum_{k=1}^{K} f_k \sum_{i=1}^{k} \frac{(i-1)P_c W}{R(M)}. \qquad (2.13)$$

The *conflict ratio*, which is defined as the ratio of the total number of locks held by txns and the total number of locks held by active txns [Moenkeberg and Weikum 1992; Weikum et al. 1994], is related to ρ as conflict ratio $= 1/(1-\rho)$ or conversely $\rho = 1 - 1/\text{conflict ratio}$.

The probability that a txn is blocked at level i is approximated by $P_b(i) = \rho^{i-1}$, $i > 1$ with $P_b(1) = 1 - \rho/(1-\rho)$. The probability of lock conflict is given by Eq. (2.1). The mean waiting time with one level of blocking can be expressed as follows by noting Eq. (2.5) for a single txn class.

$$W_1 = \frac{1}{H} \sum_{k=1}^{K} f_k \sum_{i=1}^{k} i s_k^i(\underline{\bar{M}}_a) \cdot \left[\sum_{j=1}^{k} s_k^j(\bar{M}_a) + (k-i) P_c W \right]. \qquad (2.14)$$

The normalization constant is $H = \sum_{k=1}^{K} f_k \sum_{i=1}^{k} i s_k^i(\bar{M}_a)$. The term in the brackets is the mean waiting time incurred when a txn has a lock conflict with an active txn in C_k in its ith processing step (we have assumed that txn steps have an exponential distribution).

The mean waiting time of a txn blocked at level i is approximated by $W_i = (i - 0.5)W_i$, $i > 1$ as before. Similarly to Eq. (2.10) we have

$$W = W_1(1 + 0.5\rho + 1.5\rho^2 + 2.5\rho^3 + \ldots). \quad (2.15)$$

Since $\rho < 1$, a closed-form approximation for the preceding series can be obtained assuming that it is infinite; we have

$$W = W_1\left[1 + \frac{0.5\rho(1+\rho)}{(1-\rho)^2}\right]. \quad (2.16)$$

Multiplying both sides of the equation by $K_1P_c/R(M)$ yields

$$\beta = \alpha\left[1 + \frac{0.5\rho(1+\rho)}{(1-\rho)^2}\right]. \quad (2.17)$$

Note that in addition to α, the parameter ρ is required for the analysis in this case.

To simplify the discussion, we assume that the durations of txn steps are given and are independent of $\bar{M}_a$. An iterative solution is required in this case, which proceeds as follows. Initialize $W = 0$; compute $R(M)$, $\bar{L}_a$, $\bar{L}_b$, and ρ; compute W from Eq. (2.16). Repeat the iteration step until convergence is attained with respect to W. The iteration converges in a few cycles and predicts system performance up to the peak throughput quite accurately.

Investigations with the numerical solution and simulation results obtained by varying the duration of the last txn step, which has the most impact on lock-holding times, lead to $0.2 < \rho < 0.3$, where the lower (resp., upper) limit is attained when the duration of the last step is very long (resp., equal to others). This range of values for ρ corresponds to $1.25 <$ conflict ratio < 1.43, which is consistent with results in Moenkeberg [1992]. Similarly to the case of txns with identical per-step processing times, $\bar{M}_a$ is maximized at $\beta \simeq 0.3$.

Factors affecting thrashing. In the case of txns with identical per-step processing times, the system load is determined by $\alpha = K_1P_cA$, where P_c increases linearly with the mean number of txns in the system provided that the effective database size remains fixed. The load in a system with different per-step processing times is similarly affected by α.

There is an inherent variability in the number of txns in an open system with Poisson arrivals, although the txn arrival rate (λ) is fixed. This may temporarily lead to $\alpha > 0.226$ and possibly thrashing. Load control can be applied by limiting the degree of txn concurrency to ($\bar{M}_{\max}$), which can be determined through analysis or simulation for a specific workload. This is not a desirable approach, since the variability in the workload processed by the system may lead to system underutilization or thrashing behavior.

One method to reduce the degree of txn concurrency is for txns to incur fewer disk accesses, since disk access times are significantly longer than CPU processing times. This can be partially accomplished by increasing the size of the database buffer to achieve a higher hit ratio, but a point of diminishing returns is soon reached. The main storage database paradigm, such as in IMS Fastpath [Gray and Reuter 1992], eliminates disk I/O altogether for smaller databases, except for logging purposes.

Given a fixed overall processing capacity, it is advantageous to have a smaller number of fast processors rather than a larger number of slow processors, since aside from multiprocessing effects (effective total processing capacity due to cache interference, storage access conflicts, and OS lock conflicts), a higher degree of txn concurrency (with a higher data-contention level) is required in the latter case to attain the same throughput.

Analytic and simulation results show that the mean number of active txns ($\bar{M}_a$) is maximized at $\beta \simeq 0.3$ at a degree of txn concurrency $\hat{M}$, regardless of whether the per-step processing times are identical. This provides a very easy-to-use paradigm for load control, except in the case of the heterogeneous database-access model in Section 2.5. In the case of an infinite-resource model, as we increase the number of txns in the system the throughput remains flat beyond $\hat{M}$, since $\bar{M}_a$ remains the same and the number of blocked txns increases. It is lock conflicts with blocked txns that cause a snowball effect leading to thrashing; that is, it is best not to increase the number of txns beyond the point at which the peak throughput is attained.

The susceptibility of a system to thrashing is affected by the variability in the txn mix in the form of txn sizes, read-only versus update txns, and so on. A system with fixed-size txns (with identical per-step processing times) may run at the critical value α^* or even higher values of α^* for a long time (i.e., large number of txn completions) before thrashing occurs, whereas a system with the geometric txn size distribution may thrash for values below α^* [Thomasian 1993]. In a simulation study reported in Thomasian [1993], the number of txn completions to thrashing decreases as the variance of txn size is increased for the same mean txn size.

2.3 Performance Analyses of Static Locking

The identities of all required locks are assumed to be known a priori in SL, which is only possible at a coarse granularity of locking (e.g., locks for relational tables [Date 1983]). The identities of requested locks depend on the txn's input parameters and the state of the database, the latter of which is not known a priori but can be determined by preexecuting txns (see Section 4). The execution of a txn with SL is started only when all locks have been acquired. The strict FCFS and nonstrict FCFS txn scheduling policies for SL are considered in Thomasian and Ryu [1983]. With strict FCFS the queue of blocked txns is scanned when new locks become available upon the completion of a txn and txns are activated in strict FCFS order; that is, a txn in the queue cannot be activated if txns preceding it in the queue have not been activated. Strict FCFS scheduling can also be implemented by allowing an arriving txn to acquire available locks and enqueueing lock requests for unavailable locks, both by an atomic action, to prevent deadlocks with other txns arriving simultaneously [Galler and Bos 1983].

A nonstrict FCFS scheduler scans blocked txns in FCFS order, but txns at the head of the queue are bypassed if their lock requests cannot be fully satisfied. The fact that there are no partial allocations of locks to txns improves performance by precluding unnecessary lock conflicts among newly arriving txns and blocked txns. An arriving txn can start execution immediately if its lock requests are satisfied, that is, if it does not have a lock conflict with currently active txns. Otherwise, this txn is enqueued in FCFS order. The starvation problem of the nonstrict FCFS policy, that a txn requesting a large number of locks may be bypassed repeatedly, can be rectified by reverting to a strict FCFS policy.

The strict FCFS policy, unlike the nonstrict policy, does not satisfy the *essential blocking* property that a txn can only be blocked by an active txn doing "useful work," leading to its commit [Franaszek and Robinson 1985]. It is not surprising therefore that a strict FCFS policy is outperformed by the latter policy as quantified in Thomasian and Ryu [1983]. Note that even better performance could be attained with *strict essential blocking* [Franaszek et al. 1992], which differs from essential blocking in that it requests locks only when they are required, that is, on demand locking.

Assuming that fine-granularity locking is possible with SL, it entails an increased lock holding time as compared to DL. *Incremental SL* is a form of DL where locks are requested in quick succession at the beginning of txn execution. It is therefore not surprising that DL outperforms incremental SL [Tay et al. 1985; Tay 1987]. More generally, DL outperforms SL for lower degrees of txn concurrency, which is due to the shorter lock-holding times and lower lock contention in DL, but it is outperformed by SL when it thrashes [Tay et al. 1985; Thomasian and Ryu 1986; Tay 1987]. This is because with a nonstrict SL method the number of txns eligible for execution increases as more txns are made available for execution; that is, the system throughput increases up to the saturation of the bottleneck resource.

There have been a large number of performance studies of atomic SL [Potier and LeBlanc 1980; Galler and Bos 1983; Thomasian and Ryu 1983; Morris and Wong 1985; Tay et al. 1985; Thomasian 1985; Tay 1987; Ryu and Thomasian 1988; Thomasian and Ryu 1989]. An analysis of SL using a hierarchical solution method appears in Potier and LeBlanc [1980], which is flawed in its probabilistic analysis of txn conflicts [Langer and Shum 1982; Thomasian 1996a]. The analyses in Thomasian and Ryu [1983] and Morris and Wong [1985], although tedious, accurately estimate the performance of SL.

The analysis in Galler and Bos [1983] uses heuristic approximations, but the results are not accurate in all cases. A simplified analysis of SL in Tay [1987] turns out to be inaccurate in a few high-contention cases considered in Morris and Wong [1985]. This analysis is extended in Thomasian and Ryu [1989] and applied to the following cases: (i) different distributions for txn processing time; (ii) multiple txn classes; (iii) shared and exclusive lock requests; and (iv) query and update txns.

2.4 Performance Analyses of Dynamic Locking

In Sections 2.1 and 2.2 we described an analytic solution method for DL. Comprehensive reviews of analytic solutions for DL also appear in Tay [1987], Ryu and Thomasian [1990a], and Thomasian and Ryu [1991]. Solution methods for DL roughly belong to one of the following categories.

—Analytic solutions based on QNMs extend the hardware-resource-contention model to incorporate lock contention [Irani and Lin 1979; Thomasian 1982]. A central-server model (see Section 1.3) is utilized to represent hardware resource contention and D pseudoservers to represent the lock-contention delays for the D database locks, which are only visited when there is a lock conflict. The mean service time at the nodes is the mean waiting time (W), which is not known a priori but can be expressed as a fraction of the mean txn residence time ($R(M)$) in the closed system with M txns (this model is also used in the following studies). An iterative solution is proposed in Thomasian [1982], since $R(M)$ is not known a priori.

—The analytic solution method in Tay et al. [1985] and Tay [1987] is based on "flow diagrams," which are defined in Section 3.3. The mean waiting time per lock conflict (W) is estimated by a detailed analysis [Tay et al. 1985; Tay 1987]. The analysis of a closed system with M txns leads to a cubic equation in the mean number of active txns ($\bar{M}_a$), with three roots in the intervals $(-\infty, M(1 - k^2\Lambda/3)$, $(M(1 - k^2\Lambda/3)$, $(M(1 - k^2\Lambda/6))$, and $(M(1 - k^2\Lambda/3)$, $+\infty)$, for $k^2\Lambda < 1.5$, where $\Lambda = M/D$. The second root is the only one of interest. The second term inside the parentheses can be expressed as $k^2\Lambda/F \simeq P_c W/R(M)$, with P_c given by Eq. (2.1). For the upper (resp., lower) bound, $W/R(M) = 1/3$ (resp., $2/3$), with an arithmetic mean $W/R(M) = 1/2$ (rather than 1/2.25 in the author's earlier works [Tay 1990]), which coincides with the approximation used in Thomasian [1982].

—The analysis in Section B in Thomasian and Ryu [1991] allows variable txn sizes and different processing times for txn steps, which are obtained from an underlying QNM. Only two levels of txn blocking with currently active txns and txns that are blocked by active txns are considered. Simulation results show that this analysis is quite accurate up to relatively high lock-contention levels, but cannot predict peak throughput in systems with infinite resources, that is, at very high levels of txn concurrency. A similar analysis is presented in Yu et al. [1993].

—An analytic solution based on a Markov chain model whose states represent the number of active txns (J) in a closed system with M txns appears in Ryu and Thomasian [1990a]. The holding time in each state is determined by analyzing the QNM of the underlying computer system model. At the completion of a step a txn requests a new lock or commits. A lock request may result in the transition $J \rightarrow J$, $J \rightarrow J - 1$, and $J \rightarrow I$, $I \geq J$ depending on whether the lock request is successful, unsuccessful with blocking, or unsuccessful with an abort to resolve the deadlock caused by its lock request (the locks released by an aborted txn may activate other txns). This analysis thus assumes that the txn causing the deadlock is aborted to resolve a deadlock, because a more sophisticated lock-conflict resolution method, which takes into account the progress made by txns, as in WDL, is much more difficult to analyze [Thomasian 1992]. A multilevel solution method is adopted that facilitates bottom-up validation and allows modifications to some levels of the analysis without affecting others (e.g., a different lock conflict model can be adopted). It is also based on very few approximations and takes into account deadlocks, unlike almost all other analyses of DL [Tay et al. 1985; Thomasian 1993].

2.5 On More Realistic Lock-Contention Models

Most performance studies of CC methods are concerned with a homogeneous database access model, as defined in Section 1.2 and analyzed in Sections 2.1 and 2.2. Furthermore, our discussion so far has been concerned with exclusive lock requests. We introduce the effective database-size paradigm for dealing with shared as well as exclusive locks and nonuniform database accesses in the context of the homogeneous model. A minor variation from this model is the distinction made between update and query txns, where the latter only request locks in shared mode [Tay 1987]. A forerunner of the TPC-C benchmark [Gray 1993] is described and simulated in Jenq et al. [1989], which represents the more realistic heterogeneous database-access model. We briefly describe this model and propose a table-driven load-control method for it, since load-control methods for the homogeneous database access model are ineffective in this case [Thomasian 1996b]. We next discuss the issue of characterizing txn classes through inspection of txn code or monitoring txn execution. Finally, we list some possible shortcomings of the standard locking models in the context of relational databases.

Shared and exclusive locks and nonuniform database accesses. The effective database size paradigm (EDSP) provides an elegant approach for dealing with shared and exclusive lock requests and nonuniform database accesses in the case of the GW method [Tay et al. 1985; Tay 1987]. It is shown that shared and exclusive lock requests to a database of size D can be replaced with exclusive lock requests to a database of size $D_{\text{eff}} = D/(1 - s^2)$, where s denotes the fraction of lock requests that are in shared mode. In the case of the nonuniform database-access model, where a fraction b of database accesses are to a fraction c of the database, it is shown in Tay et al. [1985] and Tay [1987] that $D_{\text{eff}} = D/[b^2/c + (1 - b)^2/(1 - c)]$. EDSP in the context of restart-oriented locking methods is discussed in Section 3.2.

Heterogeneous database access model. A closed system with M txns in J txn classes and I database regions is considered. Txns in class j (C_j) have a frequency f_j and in their nth step (denoted by $C_{j,n}$) access database region i (denoted by D_i) with probability $g_{j,n,i}$. This can be represented by a bipartite graph, where one set of nodes represents txn steps and the other set the database regions; that is, $C_{j,n} \xrightarrow{g_{j,n,i}} D_i$.

The analytic solution method in Sections 2.1 and 2.2 is extended in Thomasian [1996b] to analyze this model. Validation results show this analysis to be acceptably accurate in predicting system performance up to high levels of lock contention, but not the peak throughput with an infinite resource model.

Simulation studies of two systems with four and eight txn classes and five database regions (with other parameters varied randomly over several experiments) reveal that the critical value of ρ (as defined in Section 2.2) varies in the range (0.211–0.376) in the first system and (0.245–0.376) in the other [Thomasian 1996b]. Conflict ratios are observed to be in the range (1.26–1.60) based on the experimental results reported in Weikum et al. [1994], which is consistent with these figures. In spite of the concurrence of these results, there is no guarantee that the range of values of ρ will hold in other environments. Furthermore, even if the preceding range of critical values for ρ is generally applicable, it is too wide to be useful for load control.

Consider a system whose txns belong to two disjoint sets from the viewpoint of accesses to database regions: txns in one set have a high level of lock contention, whereas the txns in the other set do not. The thrashing of txns in the first set will lead to an overall thrashing behavior in the system, because due to the frequency-based model completed txns in the second set are replaced by txns in the first set. We can prevent thrashing by restricting the number of txns in the two sets to M_1 and M_2 with $M_1 + M_2 = M$, so that the txns in the second set will not be affected by the txns in the first set.

As an example of load control based on the composition of txns in the system, consider a system with two (interfering) txn classes. The maximum throughput attained by the two classes is $T_1(\hat{M}_1, 0)$ and $T_2(0, \hat{M}_2)$. Let us assume that txns in C_1 have a higher priority and the system is guaranteed to run $T'_1(\ldots)$ ($< T_1(\hat{M}_1, 0)$) txns in C_1. Load control is achievable by providing a table $T_1(M_1, M_2)$ for various compositions of txn classes to determine the maximum degree of concurrency for txns in C_2 to guarantee the throughput for txns in C_1 [Thomasian 1996b]. This method is more flexible than the one discussed in Weikum et al. [1994], which sets limits on the degree of concurrency for txn classes and their combinations by adding one txn class at a time to the txn mix. The problem with the table-driven approach is the difficulty of generating a table for a system with a large number of txn classes. It should be noted that the number of txn classes contributing to lock contention may be rather small and it may be possible to aggregate several txn classes into one class to reduce the number of txn classes to be considered. This remains an area for further investigation.

Characterization of transaction workloads. Txns in the same txn class may have a different execution sequence based on the input data to the txn and the state of the database. For example, 98% of debit txns are processed normally, whereas 2% result in an overdraft requiring additional processing (e.g., to check credit limits, etc.). This case can be specified by two subclasses with respective frequencies. More complex txns may not lend themselves to a simple classification of their subclasses based on inspecting txn code, and the monitoring of txn execution is required for this purpose. In effect, each txn subclass is characterized by its frequency and the sequence of database calls made by the subclass is deterministic. The number of txn subclasses is determined by all possible paths from the initial to the final state(s) of the txn, which may be quite large, even if there are no cycles in its execution. The frequency of a txn subclass is the probability associated with the path. This system can be analyzed by extending the solution method in Thomasian [1996b].

Other extensions to lock-contention models. This discussion is in the context of relational DBMSs because of their popularity.

—Relational models provide locking at multiple granularity levels, intent locks, and cursor locks [Date 1983; Gray and Reuter 1992]. A better understanding of the locking behavior of txn processing systems is required for specifying realistic locking models for analysis (see e.g., Singhal and Smith [1997]).

—Determining lock-conflict probabilities with a rather complex lock-compatibility matrix, as in the case of relational DBMSs (see e.g., Date [1983] and Gray and Reuter [1992]), is a challenging problem, especially at higher lock-contention levels.

—A large number of specialized locking methods with provisions for recovery have been developed for operations on index structures [Mohan and Levine 1992]. Performance evaluation of CC methods for index structures such as B+ trees has been an area of research activity by itself. A survey of previous work and a simulation study appear in Srinivasan and Carey [1993]. Analytic methods to compare the performance of several locking methods for B* trees are presented in Johnson and Shasha [1993].

—It is postulated that an on-demand locking scheme requests one lock at a time, but this is not necessarily so. A database call may entail accesses to multiple records and the txn may only proceed after all of the requested locks are acquired.

—Most performance studies of CC methods postulate data accesses uniformly distributed over a database whose size (D) is given. In fact, only a subset of database objects is accessed at any one time, usually with nonuniform probabilities [Singhal and Smith 1997]. Given the probability of lock conflict (P_c), the database size D can be estimated using Eq. (2.1), after taking into account that a fraction s of lock requests is in shared mode.

—A possible weakness of performance studies of CC methods is that the database size (D) remains fixed as the degree of txn concurrency (M) or the txn arrival rate (λ) is increased. This is not so for some applications such as banking, as exemplified by the TPC-A or TPC-B benchmarks [Gray 1993], where the number of bank records increases in proportion to the number of its clients.

—Schedulers for txn processing systems take into account several factors, mainly based on txn response-time objectives, but also to control the lock-contention level. Briefly, an IMS txn has a type and multiple txn types may belong to a class. Txn classes are assigned to "message-processing regions" such that each region can process several txn classes and a class may be processable at several regions. There may be a different processor-priority level associated with each class in a region. The effect of txn scheduling on the level of lock contention has not been taken into account in performance studies.

2.6 Methods to Improve Locking Performance

The concurrent processing of long read-only queries for decision support applications and short update txns introduces conflicts between the two workloads, especially when the former follows a strict 2PL paradigm. One approach is to run queries at a reduced level of consistency of 1 and 2, which implies reading uncommitted data and obtaining shared locks only for the duration of the read, respectively [Date 1983; Gray and Reuter 1992]. The latter ensures cursor stability but does not guarantee repeatable reads, which requires shared locks with strict 2PL [Date 1983; Gray and Reuter 1992]. There is a significant degree of contention between cursor locks and exclusive locks, according to Singhal and Smith [1997], which results in a slowdown of query processing and wasted system resources (e.g., allocated buffer space). A large number of versioning methods have been proposed to address this issue, as reviewed in Bober and Carey [1992] and Mohan et al. [1992b]. The simulation study in Bober and Carey [1992] shows that versioning (at the record rather than page level) allows improved performance for query processing, and Mohan et al. [1992b] presents a transient versioning method in the ARIES framework [Mohan et al. 1992a].

The nested txn paradigm offers more decomposable execution units and finer-grained control over concurrency and recovery than flat txns [Moss 1985; Gray and Reuter 1992]. The decomposition of units of work into subtasks and their appropriate distribution in a computer system is a prerequisite for intratxn parallelism [Haerder and Rothermel 1993]. Multilevel txns are related to nested txns, but are more specialized [Weikum 1991; Gray and Reuter 1992]. For example, txns hold long-term record-level locks (more generally object-level locks [Weikum and Hasse 1993]), whereas page-level locks are held by subtxns for the duration of operations on records. Compensating operations for subtxns are provided for rollback recovery. An implementation of multilevel txns and a performance evaluation using synthetic benchmarks is reported in Weikum and Hasse [1993].

Ordered sharing allows a flexible lock-compatibility matrix (among shared and exclusive lock requests) as long as operations are executed in the same order as locks are acquired [Agrawal et al. 1994]. Thus it introduces restrictions on how the program realizing the txn is written; for example, the acquisition of an exclusive lock on an object should be followed immediately by its update. A txn T_2 may obtain a shared lock on an object locked in exclusive mode by T_1 (i.e., read the value written by T_1) but this will result in deferring T_2's commit to after T_1's commit. Simulation results show that there is an improvement in performance with respect to standard locking that can be ascribed to the reduced txn waiting time with respect to standard locking [Agrawal et al. 1994].

Altruistic locking allows txns to donate previously locked objects, once they are done with them but before the object is unlocked [Salem et al. 1994]. Another txn may lock a donated object, but to ensure serializability it should remain in the wake of the original txn (i.e., accesses to objects should be ordered). Cascading aborts, which are a possibility when the donated object was locked exclusively, can be prevented with strict 2PL. This makes the approach more suitable for read-only queries or long-running txns updating few records.

The proclamation-based model for cooperating txns is described in Jagadish and Shmuelli [1992], which in addition to its original motivation can be used to reduce the level of lock contention. This method is different from altruistic locking in that a txn, instead of releasing its lock on an object (that it is not going to modify again), proclaims one or more possible values for the object, which can be accessed by other txns. Txns interested in the object can proceed with their execution according to the proclaimed value(s). Note that this method has similarities to the polyvalues method in Montgomery [1978] (see Section 4.3).

Increment/decrement locks are one approach to relieve the level of lock contention for aggregate values [Gray and Reuter 1992], but there is no checking of the value, for example, that it is negative. The *escrow method* [O'Neil 1986] is a generalization of the field calls approach in IMS Fastpath [Gray and Reuter 1992], where the minimum, current, and maximum values of an aggregate variable are made available to other txns. Let us assume that a banking account with a current balance of $1,000 has a credit txn for $100 and a debit txn for $200 in progress. Then the balance is represented by ($800, $1,000, $1,100). The final balance is $900 if both txns are successful. A debit txn for $500 need not be delayed awaiting the completion or abort of either or both txns, since there are adequate funds in the account in either case.

The synchronization technique and scheduling policy described in Bayer [1986] allows simultaneous processing of a batch txn and short update txns. A "random" batch txn updates database records only once in each run (converting them from old to new records) and the updates are independent of the order in which they are carried out. There is a conflict if a short txn needs to access old and new records. Since the blocking delay is not tolerable for short txns, the batch txn may update the *required* old records and make them available to short txns after intermediate commit points [Date 1983; Bayer 1986].

Lock-holding time by long-lived txns can be reduced by using intermediate commit points according to the sagas paradigm [Garcia-Molina 1987; Gray and Reuter 1992]. A long-lived txn T is viewed as a set of subtxns $T_1, \ldots, T_n$ that are executed sequentially and can be committed individually at their completion. However, the abort of subtxn T_j results in the undoing of the updates of all preceding subtxns from a semantic point of view through compensating subtxns $C_1, \ldots C_{j-1}$. Compensating txns consult the log to determine the parameters to be used for undoing the updates of aborted txns.

A method for chopping txns into smaller txns to reduce the level of lock contention and to increase the level of txn concurrency, while preserving correctness, is presented in Shasha et al. [1995], which also reviews related works. Preserving correctness requires knowledge about the set of txns that are running concurrently with the txn to be chopped. Examples of correct and incorrect "choppings" are given in the paper and are not repeated here; suffice it to say that the critical steps that may fail should be executed first. Simulation results show a significant improvement in performance when a long-running txn (say a batch update) is chopped into smaller txns, but this is at the cost of an increased number of commits with the associated overhead [Shasha et al. 1995]. A related study that extends Farag and Ozsu [1989] and Korth and Speegle [1994] appears in Ammann et al. [1997].

Semantics-based CC methods rely upon the semantics of txns (see, e.g., Garcia-Molina [1983]) or semantics of objects [Weihl 1988; Badrinath and Ramamithram 1992]. Txn semantics is utilized in Garcia-Molina [1983] and Cellary et al. [1988], where txns are classified into types and a compatibility set is associated with different types. A semantics-based CC method for objects is based on commutativity of operations. Recoverability of operations is an extension of this concept, which allows an invoked operation to proceed when it is recoverable with respect to an uncommitted operation [Badrinath and Ramamithram 1992]. Various operations on stacks and tables belong to this category. The two methods based on commutativity of operations presented in Weihl [1988] differ in that one method uses intention lists and the other uses undo logs. This work is extended in Lynch [1994]. Object-oriented DBMSs allow a higher degree of txn concurrency than provided by simple locking for operations on abstract data types [Skarra and Zdonik 1989; Ozsu 1994]. A more detailed discussion of these topics appears in Ramamithram and Chrisanthis [1996].

3. RESTART-ORIENTED LOCKING METHODS

Restart-oriented locking methods combine blocking and aborts (followed by system-initiated restarts) to cope with the performance limitations of standard locking. We first describe restart-oriented locking methods, various options for restarting aborted txns, and the issue of resampling of lock requests for restarted txns with respect to the original txns. Section 3.2 provides a performance comparison of restart-oriented locking methods based on simulation results reported in Thomasian [1997a]. The modeling of shared locks, in addition to exclusive locks and hot-spots, is also discussed in this section. Finally, in Section 3.3 we review analytic studies of restart-oriented locking methods and analyze a representative method.

3.1 Description of Restart-Oriented Locking Methods

The wait depth is the distance (d) of a blocked txn from active txns at the root nodes of the respective acyclic WFG. Only exclusive locks are considered initially to simplify the discussion, in which case WFGs are directed trees. A concise specification of some well-known restart-oriented locking methods is as follows:

—According to the no-waiting (NW) [Tay et al. 1985b] or the immediate restart method [Agrawal et al. 1987a] a txn T_A that has a lock conflict with a txn T_B is aborted. The WFG $T_A \rightarrow T_B$ is temporarily formed and reduced. The wait depth is $d = 0$.

—The asymmetric running priority (RPA) method [Franaszek and Robinson 1985] aborts T_B in the WFG $T_A \rightarrow T_B \rightarrow T_C$ when T_A becomes blocked by T_B, which is already blocked by T_C. This action is expected to improve system performance, since it partially fulfills the strict essential blocking property [Franaszek et al. 1992] (see Section 2.3).
It is possible that T_B, which was active at the time T_A became blocked by it, may become blocked by T_C at a later time. When T_C is active, the Symmetric RP (RPS) method [Franaszek et al. 1992] guarantees that the wait depth does not exceed one by aborting a txn such as T_B, which is blocking other txns when it encounters a lock conflict. The wait depth is maintained at $d = 1$ in this case.[7]
Adaptive methods are desirable in that they adjust the wasted processing due to txn aborts to match the excess processing capacity of the system. An adaptive RP method only aborts a txn at $d = 1$ blocking $b \geq 1$ txns, where the txn breadth b is the number of txns blocked by it [Franaszek et al. 1991a] (this adaptive method is used in the context of a two-phase processing method in Section 4.4). Wasted processing can be reduced by increasing the value of b or by aborting a txn only when its wait depth $d > 1$ [Franaszek et al. 1992]. It is also meaningful to consider a combination of rules based on the wait depth and breadth (e.g., (i) if $d > 1$ and $b \geq 1$, and (ii) if $d = 1$ and $b > 1$).

—The asymmetric cautious waiting (CWA) method [Hsu and Zhang 1992] aborts T_A when it is blocked by T_B, which is itself blocked by T_C as in

[7] There are two alternatives to ensure this. The first method checks whether a txn T_A encountering a lock conflict is blocking other txns; if so, it is aborted. Next it is checked whether T_B blocking T_A is blocked itself, and if T_B is in the blocked state, it is aborted. Simulation results have shown that changing the ordering of these two tests has little effect on overall performance.

$T_A \rightarrow T_B \rightarrow T_C$. The symmetric CW (CWS) method first checks if T_A is blocking other txns, as in $(T_X, T_Y, \ldots, T_Z) \rightarrow T_A \rightarrow T_B$, and aborts them when T_A becomes blocked. The wait depth is maintained at $d = 1$ with CWS.
Symmetric and asymmetric RP and CW are deadlock-free [Franaszek and Robinson 1985; Hsu and Zhang 1992].

—The WDL(d), $d \geq 1$ family of methods limits the wait depth of blocked txns to d, taking into account txn progress in deciding which txn to abort to achieve this goal [Franaszek et al. 1992]. They are a subset of wait-depth-limited methods, which only limit the wait depth (such as RPS or CWS). The WDL(1) or WDL method is mainly of interest [Franaszek et al. 1992], since it attains high performance at the cost of relatively little wasted processing. Conflicts are resolved by comparing the length of the txns involved in the conflict. The length $L(T_A)$ of a txn T_A is a function of the progress made by the txn (e.g., the number of locks held by T_A [Franaszek et al. 1992]). The modified WDL (MWDL) method [Thomasian 1992] is based on two slightly different rules from WDL [Franaszek et al. 1992] such that the comparison of txn lengths is always confined to two txns at a time.
Consider a lock request by T_A that causes a lock conflict with T_B resulting in a transient WFG $[(T_X, T_Y, \ldots, T_Z) \rightarrow] T_A \rightarrow T_B [\rightarrow T_C]$. The following rules are applied when a lock conflict occurs with the MWDL method.

(1) If T_A, which is blocking some other txns, has a lock conflict with T_B, and then if $L(T_A) < L(T_B)$, abort T_A, else abort T_B.[8]

WDL makes the comparison $L(T_A) < \max(L(T_B), L(T_X), \ldots, L(T_Z))$ when T_A is blocking txns $(T_X, T_Y, \ldots, T_Z)$ [Franaszek et al. 1992].

(2) If T_A, which is not blocking any other txns, has a lock conflict with T_B, which is itself blocked by an active txn (T_C), if $L(T_B) \leq L(T_C)$, then abort T_B, else abort T_C.
WDL makes the comparison $L(T_B) \leq \max(L(T_A), L(T_C))$ [Franaszek et al. 1992].
Although the WDL family of methods allows wait depths greater than one, a wait depth of one yields improved performance [Franaszek et al. 1992; Thomasian 1997a] without requiring excessive extra processing.

—The wound-wait (WW) method [Rosenkrantz et al. 1978] blocks txn T_A requesting a lock held by T_B if T_A is not older than T_B; otherwise T_B is aborted. A variant of WW, similarly to RPA, wounds T_B only if it is blocked (at the site where the conflict occurred in a distributed database).

—The wait-die (WD) method [Rosenkrantz et al. 1978] allows a younger txn to wait if it is blocked by an older txn; otherwise the txn encountering a lock conflict is aborted.
Txn age is determined by the arrival-time timestamp. Both WW and WD methods are deadlock-free [Rosenkrantz et al. 1978], but similarly to the CWA and RPA methods allow an unlimited wait depth. The WW and WD methods are intended as low-cost deadlock-prevention mechanisms in distributed database environments from the viewpoint of the number of messages involved. The WW outperforms WD according to the simulation results in [Agrawal et al. 1987b], which is the reason it is not considered in Section 3.2.

NW, CW, WD, and the method in Chesnais et al. [1983] (see Section 3.3) are *nonpreemptive methods* [Hsu and Zhang 1992], in that the txn encounter-

[8] We could have used $L(T_A) > L(T_B)$ instead of $L(T_A) \geq L(T_B)$ for the else condition and a tie-breaking rule for equality (e.g., abort the younger txn). This extra complication is not justified, since it is not expected to result in an improvement in performance.

ing a lock conflict is aborted, whereas RP and WW are *preemptive methods* according to this definition. A classification is also possible based on txn attributes used in determining the txn to be aborted (if any). Txn timestamps in the case of the WW and WD methods are *static attributes* that are assigned at the beginning of txn execution (txn timestamps are reassigned for each instance of txn execution in the case of the TSO method). WDL uses a *dynamic attribute* (i.e., the number of locks currently held by a txn).

Transaction aborts and restarts. We distinguish between these two terms, which have been used interchangeably. Txn abort occurs first and a txn releases its locks and stops its execution or continues executing in virtual-execution mode (see Section 4.2). Aborted txns whose execution is stopped and txns in virtual-execution mode completing their execution are automatically restarted by the system. Cyclic restarts or livelocks need to be prevented: (i) to reduce the wasted processing incurred in this manner, and (ii) to ensure txn completion within a finite time interval. The following methods to prevent cyclic restarts are complementary to those in the introduction to Section 2.

(1) Restart waiting can be easily implemented in a centralized system by delaying the restart of an aborted txn until *all* txns that have conflicted with it are completed [Ryu and Thomasian 1990b; Franaszek et al. 1992].
(2) Random delays or conflict avoidance delays [Agrawal 1987a,b; Tay 1987], which are introduced before a txn is restarted after its abort, do not guarantee that cyclic restarts will be prevented. The drawback of applying this method is the difficulty of determining the duration of random delays, especially in the case of variable-size txns.
(3) Immediate restarts are possible in a system with a backlog of txns, such that an aborted txn is set aside and is replaced by a new txn with a different script [Tay et al. 1985; Agrawal et al. 1987a], which is defined as the sequence of objects accessed by the txn. Another interpretation is that the restarted txn requests a different set of locks from its prior execution; that is, the txn does not exhibit access invariance [Franaszek et al. 1990, 1992]. This is referred to as *lock resampling* or *fake restarts*, as opposed to *no lock resampling*.

The restart waiting-no lock resampling combination of options is mainly of interest. No lock resampling occurs with access invariance across successive executions of a restarted txn and is expected not to affect the level of lock contention in the system when txn restarts occur, although this is not so for lock resampling which favors methods introducing more txn restarts. The restart waiting method is straightforward to implement and allows a fair comparison.

Lock resampling is an inherent shortcoming of the analytic solution methods that results in overestimating system performance by an extent that depends on the frequency of txn restarts. This effect is quantified in the case of the NW method in Agrawal et al. [1987a]. Resampling also applies to locking modes, the class of restarted txns, and txn processing times. Resampling results in an unfair advantage to methods with more frequent restarts, since this creates a bias of the completed txns in the system towards txns with the characteristics: (i) a higher fraction of shared lock requests, (ii) fewer accesses to hot-spots, and (iii) smaller txn sizes.

3.2 Performance Comparison of Restart-Oriented Locking Methods

There have been several simulation studies of restart-oriented locking methods, most notably Agrawal et al. [1987b] and Franaszek et al. [1992]. We summarize here the results of a simulation study with the restart waiting-no lock resampling options (justified in the previous section) and exclusive lock requests to compare the performance of eight methods: GW, NW, CWS, RPS and RPA, WDL, MWDL, and WW [Thomasian 1997]. The simulation parameters are: database size $D = 16{,}384$, txn size $K = 16$, txn steps have identical exponential distributions. The number of txns is varied to obtain the peak throughput for different hardware resource contention levels, which is attained by varying the number of processors in the system (P).

—GW outperforms other methods for smaller values of P, but achieves its maximum throughput at $\hat{M} \simeq 78$ with $\bar{M}_a = 55$ active txns; that is, GW can utilize at most $P = 55$ processors. Note also that there is little difference in the peak throughput attained by different methods at $P = 50$, since the level of lock contention is low.

—NW is outperformed by GW at $P = 50$, but NW outperforms GW by almost 20% at $P = 100$ and its throughput continues to increase up to $P = 250$. NW with the immediate restart-lock resampling option is susceptible to thrashing due to repeated restarts.

—CWS introduces a significant improvement in performance over NW that is due to the reduction in wasted processing. This is also the case for the CWA method [Hsu and Zhang 1992]. Simulation results show that the CWS and CWA methods have a comparable performance (differing by a few percentage points) and as would be expected CWS outperforms (is outperformed by) CWA for small (resp., large) values of P, since CWA introduces fewer aborts than CWS.

—The RPS method outperforms CWS in all cases, which is attributable to the fact that aborting a blocked txn in the case of RPS results in an increase in the number of active txns from one active txn in $T_A \rightarrow T_B \rightarrow T_C$ to two active when T_B is aborted and is set aside, whereas an aborted txn in the case of CWS just starts restart waiting, since an immediate restart will lead to repeated conflicts and aborts [Hsu and Zhang 1992].
For smaller values of P, RPA outperforms not only RPS but also the WDL and MWDL methods, which is attributable to the fact that RPA results in less wasted processing than the other methods, such that it can maximize the useful processing in the system.

—The WDL [Franaszek et al. 1992] and the MWDL [Thomasian 1992] methods outperform others (including RPA) at higher processing capacities. The peak throughput attained by these two methods is almost a factor of four higher than that attained by GW and 20% higher than RPA. These results concur with simulation results in Fanaszek et al. [1992]. It is interesting to note that MWDL, in spite of its relative simplicity with respect to WDL, attains a performance very close to it.

—WW, although outperforming GW for higher values of P, is outperformed by all other methods except NW. The fact that txns are blocked in timestamp order results in very little drop in peak throughput, once it is attained. In effect, unnecessary restarts are avoided and the restart-waiting option is not required in this case.

The method with the maximum effective throughput for a given P maximizes the number of processors doing useful work (say P_u). Thus for intermediate values of P, RPA outperforms WDL, since it attains a higher P_u. However, as P is increased, WDL and MWDL utilize more processors (up to $P = 500$) and attain a higher P_u than RPA [Thomasian 1993].

A simulation with a synthetic workload and a lock trace is used in Weikum et al. [1994] to compare the performance of several load-control methods, especially those based on conflict ratios (see Section 2.2). WDL is shown to have a superior performance in all cases but one, which is an accelerated synthetic workload. The main reason for the success of WDL according to this study is its smaller lock-waiting time with respect to other methods. Note that restart waiting in the case of WDL serves the role of load control.

Synthesizing methods that will maximize system throughput for a given set of system parameters remains an open problem. Guidelines for this purpose are as follows.

(1) From the viewpoint of data contention, it should conform with the essential blocking property [Franaszek and Robinson 1985] and minimize wasted lock-holding time, which affects system performance when lock utilizations are high (e.g., locks on hot-spots).[9]

(2) From the viewpoint of hardware resource contention it should minimize wasted processing by associating a higher priority level with txns that have acquired more processing than others [Franaszek et al. 1992] and exhibit adaptiveness to system load, for example, by reducing the number of txn aborts when the system is overloaded [Franaszek et al. 1991a].

(3) From the viewpoint of reducing txn response times, care should be taken not to abort txns that are near completion but have not yet entered the commit phase. This entails a reduction in lock-holding time and wasted processing, which are both almost doubled when a txn near its completion is aborted. The progress made by a txn towards its completion can be deduced from its type in some cases (a system with canned txn classes). The performance of a system that takes into account txn progress with respect to its completion in deciding which txns to abort requires further investigation.

Adaptive methods balance the reduction in lock contention with wasted processing. Minimizing the wasted processing up to a point where the bottleneck resource in the system (e.g., the processors) is almost fully utilized is a good strategy, since this tends to maximize the useful processing in the system. This was verified through simulation of the two-phase processing method in Franaszek et al. [1991a] (see Section 4.3), which is based on the adaptive RP method in Section 3.1.

Given that the future lock requests are known a priori, an "optimal" policy can be used to determine the best possible performance, for example, the minimum mean response time for a given throughput, against which other methods can be gauged. Unfortunately, an optimal policy for this purpose is computationally very expensive. In contrast, the optimal policy for minimizing the miss ratio in paging virtual-memory systems is to simply replace the page referenced furthest in the future, provided the trace of page requests is known.

The performance of methods based on OCC and access invariance is compared to the WDL method in Section 4.4.

The effect of shared locks and hot-spots on performance. It is important to determine if the relative performance of the restart-oriented locking methods also holds in the presence of shared as well as exclusive locks and hot-spots. A straightforward extension of RPA with shared locks restarts all txns holding the requested lock in shared mode if at least one of them needs to be aborted because it is blocked. This method outperforms the GW method in a system with infinite resources according to the simulation results in Thomasian [1993], but this may not be true in a system with limited hardware resources.

EDSP (introduced in Section 2.4) is

[9] To see why this is so, consider the following example. The throughput of a system with a single hot-spot can be maximized by minimizing the wasted lock holding time on the hot-spot. Thus the txn holding this lock cannot be aborted or blocked; that is, txns that get in its way (by holding locks required by it) are aborted.

expected to be applicable to wait-depth-limited methods, since they maintain a subset of the WFG for the GW method, but this is not always so. For example, according to Corollary 5.2 in Tay et al. [1985b], EDSP for shared and exclusive locks for NW with lock resampling holds only for sufficiently low levels of lock contention. This problem is considered further in Hsu and Zhang [1995] in the context of NW and CWA, which are "equal-chance abort policies"; that is, the probability of txn abort upon a lock request is independent of its current state. The data-flow balance principle is used in the context of the hot-spot database access model to show that the effective database size (D_{eff}) is an increasing function of M and that EDSP underestimates D_{eff} (overestimates the lock conflict probability) by estimating D_{eff} at $M = 0$. This is because EDSP does not take into account the lock-resampling effect. This analysis is of limited applicability, since it deals with the less realistic lock-resampling option.

It follows from simulation results in Thomasian [1993] that EDSP provides an acceptably accurate approximation for the no-lock-resampling option, including no resampling of locking modes. For example, in the case of the NW method with frequent aborts, simulation results obtained by applying EDSP to shared and exclusive locks are indistinguishable from the results for the original model; also, very close results are obtained with the hot-spot model. From these experiments, it is conjectured that EDSP applies approximately to the NW and CWA methods for which the txn making the lock request is aborted. This is attributed to the fact that the expected mix of the modes of locks held by txns in the system is affected negligibly in this case. However, there is the second-order effect that a txn making an exclusive lock request has a higher probability of lock conflict than when it makes a shared lock request.

In the case of RPA, it is observed from simulation results that the throughput obtained by applying EDSP to the hot-spot model matches the original model quite well. The peak system throughput is typically overestimated by 5%. EDSP is less accurate in the case of RPA when applied to a system with shared and exclusive locks. This is attributable to the fact that the composition of txns in the system is affected when txns are aborted as a result of a lock conflict (as in the case of multiple txns holding a lock in shared mode, which is requested in exclusive mode, being aborted because one of them is blocked).

3.3 Performance Analysis of the Running Priority Method

We briefly review earlier analytic studies of restart-oriented locking methods and outline the analysis of a nontrivial wait-depth-limited method by analyzing the performance of the RPS method.

Review of earlier analytic studies. A method according to which a txn encountering a lock conflict repeats its request L times before it is aborted is described in Chesnais et al. [1983]. A Markov-chain model representing the progress of a single txn is used to analyze system performance. Numerical results with the immediate restart-lock resampling option and the infinite-resource assumption lead to the conclusion that the best performance is attained at $L = 0$, that is, the NW method. Comprehensive studies of NW and GW using flow diagrams appear in Tay et al. [1985b] and Tay et al. [1985a], respectively (see also Tay [1987]). It is shown in Tay et al. [1985a] that the NW method outperforms GW when there are infinite resources, but at lower degrees of txn concurrency NW is outperformed by GW (by at most 5%). Another study [Ryu and Thomasian 1990b] concludes that GW outperforms NW in its nonthrashing region, unless there are infinite resources and the immediate restart-lock resampling option is in effect. The flow-diagram method of

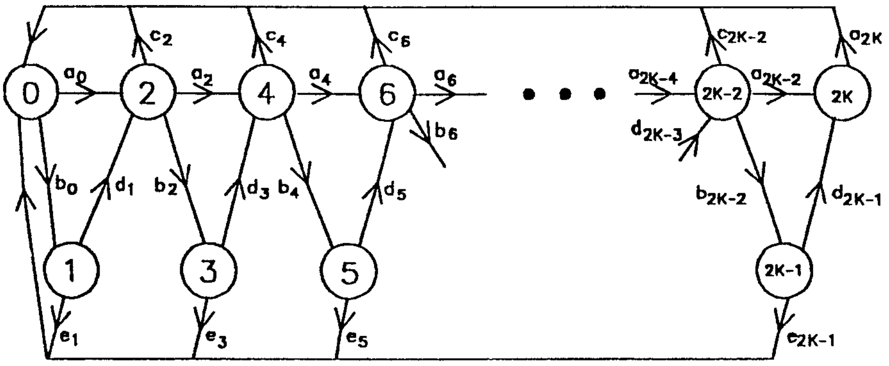

Figure 2. Markov chain for wait-depth-limited policies.

Tay [1987] is adapted to the analysis of the CWA method in Hsu and Zhang [1992].

Flow diagrams, semi-Markov chains, and Markov Chains. The analysis of restart-oriented locking methods is simplified by considering the execution of a single txn as affected by the remaining txns in the system from the hardware- and data-resource-contention viewpoint. The flow diagram or semi-Markov chain [Kleinrock 1975] depicting the execution steps of a txn T of size K is shown in Figure 2. S_{2i}, $0 \le i \le K$ (resp., S_{2i+1}, $0 \le i \le K - 1$) correspond to active (resp., blocked) states. The transition rates among the states are derived in the following.

Semi-Markov chains allow a general distribution for state holding time, whereas Markov chains stipulate an exponential distribution [Kleinrock 1975]. The use of semi-Markov chains in the analysis of the GW, NW, and CWA methods in Tay et al. [1985a,b] and Hsu and Zhang [1992], respectively, is possible because all transitions from the active state occur at the time the txn completes an execution step and makes a lock request, and only one transition is possible from the blocked state when the txn is unblocked due to the release of the requested lock. RPS, RPA, and WDL methods allow a txn to be aborted in the blocked state; that is, there are two transitions from this state, and the distribution of waiting time is required for the analysis. This is expensive computationally, as explained in Thomasian [1995a]. We therefore assume that waiting times are exponentially distributed. The assumption that the txn abort process is Poisson, which is also made in analyzing the OCC method in Section 4.4, leads to a low-cost solution for the semi-Markov chain model. Since the distribution of the processing time of txn steps has little effect on performance, to simplify the analysis further we assume that the processing times of txn steps are exponentially distributed, so that a Markov chain analysis is possible.

The state equilibrium equations for the Markov chain can be solved easily to obtain the steady-state probabilities π_i, $0 \le i \le 2K$ [Kleinrock 1975]. The mean number of visits to S_i (v_i) can be similarly obtained by noting that $v_{2K} = 1$. Given that h_i denotes the mean holding time at S_i, then $\pi_i = v_i h_i / \sum_{j=0}^{2K} v_j h_j$, $0 \le i \le 2K$. Note that the mean number of txn aborts is $v_0 - 1$.

Analysis of the RPS method with fixed-size transactions. This analysis is intended to illustrate a general solution method based on the progress made by a single txn. The state transition rates of the Markov chain according to the RPS method are given in the following (see Figure 2). Variables required for analyzing the Markov chain are derived later.

(1) $S_{2j} \to S_{2j+2}$ with rate a_{2j}, $0 \le j \le K - 1$ designates successful lock requests, and $S_{2K} \to S_0$ with rate $a_{2K} = \mu_K$ corresponds to the completion of a txn. The probability that a txn T encountering a lock conflict at S_{2j} is blocking another txn is given by Q_j. Let P_a (resp., P_b) denote the probability of lock conflict with an active (resp., blocked) txn, with $P_c = P_a + P_b$. We have $a_{2j} = [(1 - P_c) + P_b(1 - Q_j)]\mu_j = (1 - P_a - P_bQ_j)\mu_j$, $0 \le j \le K - 1$. P_a, P_b, and Q_j are derived in the following.

(2) $S_{2j} \to S_{2j+1}$ with rate b_{2j}, $1 \le j \le K - 1$ designates unsuccessful lock requests leading to txn blocking. T is blocked only if it is not blocking any other txns, and hence $b_{2j} = P_a(1 - Q_j)\mu_j$, $0 \le j \le K - 1$.

(3) $S_{2j} \to S_0$ with rate $c_{2j} = P_cQ_j\mu_j$, $1 \le j \le K - 1$ corresponds to txn aborts.

(4) $S_{2j+1} \to S_{2j+2}$ with rate d_{2j+1}, $0 \le j \le K - 1$. The waiting time for the acquisition of a lock (due to completion or abort of the txn holding the lock) is approximated by an exponential distribution $W(t) = 1 - e^{-\nu t}$, $t \ge 0$, with $\nu = 1/W$.

(5) $S_{2j+1} \to S_0$ with rate e_{2j+1}, $0 \le j \le K$ designates the abort of T, which occurs when T', which is not blocking other txns, requests a lock held by T. We assume that the process according to which T is aborted at S_{2j+1} is Poisson with rate ω_j (see Eq. (3.4)).

The probability of lock conflict with active (resp., blocked) txns is $P_a \simeq (M - 1)\bar{L}_a/D$ (resp., $P_b \simeq (M - 1)L_b/D$), where $\bar{L}_a = \sum_{j=1}^{K} j\pi_{2j}$ (resp., $\bar{L}_b = \sum_{j=1}^{K-1} j\pi_{2j+1}$) is the mean number of locks held by active (resp., blocked) txns. The fraction of blocked txns in the system is given by $\beta(= \sum_{j=1}^{K}\pi_{2j-1})$. As in the case of standard locking (see Section 2.1), the mean number of active and blocked txns is given by $\bar{M}_a = M(1 - \beta)$ and $\bar{M}_b = M\beta$, respectively. The mean number of locks held per txn is $\bar{L} = \bar{L}_a + \bar{L}_b$ and hence $P_c \simeq (M - 1)\bar{L}/D$. The probability that an active txn T at S_{2j} is blocking at least one of the $\bar{M}_b$ blocked txns in the system is approximated by

$$Q_j = 1 - [1 - j/(M\bar{L}_a)]^{\bar{M}_b}, \qquad 1 \le j \le K, \tag{3.1}$$

where $j/(M\bar{L}_a)$ is the probability that another txn is blocked by the target txn at S_{2j} (the denominator denotes the mean number of locks held by active txns).

The mean waiting time W_j due to a lock conflict with an active txn at S_{2j} is $W_j = \sum_{l>2i}^{2K} \nu'_l h_l$. Visit ratios with respect to S_{2j} are obtained as follows: $v'_{2j} = 1$, $v'_{2i+1} = b_{2i}v'_{2i}$, and $v'_{2i+2} = (a_{2i} + b_{2i}d_{2i+1})v'_{2i}$, $j \le i \le K$. This takes into account lock acquisition by txn completion or abort. The mean holding time in S_l is $h_l = s_i = 1/\mu_i$, when $l = 2i$, $j \le i \le K$. The residual processing time in S_{2j} equals the processing time in that step, since the per-step processing times are exponentially distributed [Kleinrock 1975]. The mean delay in the blocked state S_l is the expected value of the minimum of two exponential distributions $h_l = 1/(\omega_i + \nu)$ with $l = 2i + 1$, $j \le i \le K - 1$ [Kleinrock 1975]. The mean waiting time is given by

$$W = \sum_{j=1}^{K} q_j W_j, \tag{3.2}$$

where q_j is the probability of lock conflict with an active txn at S_{2j}. The probability q_j equals the time-space product of the number of locks held at S_{2j} and their holding time: $q_j = j\pi_{2j}/H$, where $H = \sum_{j=1}^{K} j\pi_{2j}$ is a normalization constant. The rate of lock requests by the other txns in the system is

$$\lambda = \left(1 - \frac{1}{M}\right)T(M) \sum_{l=0}^{K-1} v_{2l}(1 - Q_l). \tag{3.3}$$

The summation takes into account the increase in the rate of requested locks due to txn restarts and the fact that lock conflicts due to txns that are blocking other txns do not have an effect, since these txns are aborted upon lock conflict.

The rate at which a txn at S_{2j+1} is aborted is proportional to the number of locks that it holds:

$$\omega_j = j\lambda/D, \quad 0 \le j \le K - 1. \quad (3.4)$$

The mean txn response time is $R(M) = \sum_{j=0}^{K} v_{2j}s_j + \sum_{j=0}^{K-1} v_{2j+1}/(\nu + \omega_j)$. Txn throughput is given by $T(M) = M/R(M)$ or alternatively by $T(M) = M\mu_K\pi_{2K}$. Some of the parameters required for the analysis are not known a priori; hence an iterative solution is required, which tends to converge in a few cycles.

Validation with infinite resources shows the analysis of the RPS method to be quite accurate up to very high lock contention levels, but to underestimate peak throughput by 8%, which is partially attributable to the fact that the analysis does not take into account the possibility of multiple txns being unblocked when another txn is aborted [Thomasian 1995a].

Outline of the analysis of GW and NW methods. In the case of the NW method, there are only two transition types: $a_{2j} = \mu_j(1 - P_a)$, $0 \le j \le K - 1$ and $c_{2j-1} = \mu_j P_a$, $0 \le j \le K$. It can be easily shown that $\pi_{2j} = \pi_0(1 - P_a)^j$, $1 \le j \le K$. Multiplying both sides of this equation by M yields the mean number of txns in different states, as obtained by the analysis of flow diagrams in Tay et al. [1985b] and Tay [1987]. The analysis in this case is tantamount to solving a polynomial equation in $q = 1 - P_a$, which can be derived from the last equation noting that $P_a \simeq \bar{N}/D$, where $\bar{N} = M \sum_{j=1}^{K} j\pi_j$ is the mean number of locks held in the system. The polynomial has a unique root in the range (0,1) [Tay 1987].

In the case of GW there are three transition types, provided that as in Tay et al. [1985a] we ignore the effect of deadlocks: $a_{2j} = \mu_j(1 - P_c)$, $0 \le j \le K - 1$, $b_{2j} = \mu_j P_c$, $0 \le j \le K$, and $d_{2j-1} = \nu = 1/W$, $1 \le j \le K$. The state equilibrium equations are given by $\pi_{2j-1} = (\mu P_c/\nu)\pi_{2j-2}$, $1 \le j \le K$ and $\pi_{2j} = \pi_0 = [1 + K(1 + \mu P_c/\nu)]^{-1}$, $1 \le j \le K$. Active states and blocked states have equal probabilities, since the probability of a txn encountering a lock conflict is independent of its step and the possibility of abort to resolve deadlocks is ignored. System performance in this case depends on P_c and $\nu = 1/W$. Multiple levels of txn blocking need to be considered to estimate W in this case [Tay et al. 1985a; Thomasian and Ryu 1991; Thomasian 1993c].

Analysis of RPS with variable transaction sizes. The analysis of the frequency-based model of the NW method in Tay et al. [1985b] and Tay et al. [1987] uses different transition points from a single flow-diagram to denote txn completions. This analysis does not ensure conservation of txn class frequencies; that is, the fraction of txns completed by the system differs from the original frequencies in a manner favoring shorter txns, which have a higher probability of success than longer txns. The analysis for a single txn class can be extended to multiple classes with the frequency-based model by using a separate Markov chain per class in the analysis, as is done in Thomasian [1992, 1995a]. A system with a given number of txns in two classes is analyzed by a simple extension of the analysis for a single txn class in Tay et al. [1985b] and Tay [1987].

4. TWO-PHASE PROCESSING METHODS AND ACCESS INVARIANCE

The first phase of txn execution may lead to its commit, but even if it is not successful its execution to completion prefetches data from disk for the second execution phase of the txn. Two-phase processing methods increase the chances of successfully re-executing txns, if necessary, by allowing a shorter processing time for txns in the second phase and a lower effective degree of txn concurrency in this and further phases, provided that the database buffer is large enough to retain pages accessed by txns and access invariance prevails [Franaszek et al. 1992]. Access invariance can be at the logical or physical level. In the first case a txn accesses the same logical objects (e.g., records) and in the second case the same physical objects (e.g., blocks) are accessed upon re-execution [Franaszek et al. 1990, 1992]. OCC methods are well suited for two-phase processing, as is shown in the following discussion. Multiphase processing is possible with OCC, since txn completion in the second execution phase is not guaranteed.

The restart-oriented methods discussed in Section 3 also benefit from access invariance to some degree; that is, a restarted txn can usually access pages from the database buffer resulting from a txn's prior execution (up to the point of the txn's abort), but unlike two-phase processing no attempt is made to continue the execution of an aborted txn.

Section 4.1 is a brief introduction to OCC and its validation options. Mechanisms for two-phase txn processing are described in Section 4.2, followed in Section 4.3 by a description of txn scheduling methods and their relative performance with respect to each other and other methods. In Section 4.4 we outline analytic solution methods for OCC.

4.1 Optimistic Concurrency Control

Optimistic CC (OCC) is a major alternative to locking [Kung and Robinson 1981; Robinson 1982a], but is less suitable than locking in meeting the requirements of DBMSs for high-performance txn processing [Haerder 1984; Mohan 1992]. However, there have been many prototyping efforts (see Thomasian and Rahm [1990] for a partial list), numerous proposals for improved algorithms [Menasce and Nakanishi 1982; Robinson 1984; Haerder 1984; Franaszek et al. 1992], and several analyses of OCC performance [Menasce and Nakanishi 1982; Moenkeberg and Weikum 1985; Ryu and Thomasian 1987].

The execution of a txn with OCC comprises the following phases [Kung and Robinson 1981].

(1) During the read phase txns access database objects, possibly updating a subset of these objects. A "clean" or committed copy of requested objects is made available to txns from the database buffer in this phase, and dirty copies of the same objects may exist in the private workspace of other txns. Reciprocally, the updates (prewrites) of txns during the read phase do not affect the database buffer, but only the txn's private workspace.

(2) The validation phase is required to ensure serializability by checking for data conflicts as described in the following. Data conflicts are resolved by aborting txns that fail their validation.

(3) A read-only txn is considered completed after a successful validation, and a txn that has updated some objects during the write phase initiates commit processing by writing the log records onto disk and then externalizes updated objects to the database buffer.

The validation method described in Kung and Robinson [1981] has the shortcoming of unnecessarily aborting txns, which is rectified in Robinson [1984]. The implementation described in Menasce and Nakanishi [1982] and Ryu and Thomasian [1987] requires the association with "active" database objects of a timestamp $t(O)$ that indicates the time at which object O was last updated. Let $t(O, T)$ denote the time that object O was read by tnx T. The following two rules are required for the correct execution of a txn T.

Read O. Copy object O into T's local workspace. In case there is no timestamp associated with an object, its timestamp is set to the current time $t(O) = Clock$. Also set $t(O, t) = Clock$.

Validate T. For all objects O accessed by the txn, check in a critical section whether $t(O, T) \geq t(O)$:

—If the condition is not true for *any* object, then abort T.

—Otherwise commit T and externalize all objects O in the write set after setting $t(O) = Clock$.

Validation may be carried out in a critical section serializing the validation process, which may become a bottleneck at high txn volumes. Performance can be improved by acquiring exclusive locks on all accessed objects by an atomic action (static locking with strict FCFS policy; see Section 2.3).

The OCC method can append access entries (similarly to locks in standard locking) to linked lists associated with hash classes identifying accessed objects. Access entries are used at txn commit time in identifying conflicted txns and marking them as "injured." According to the kill or broadcast commit method, a txn is aborted "right away," but in fact txn abort is usually deferred until its next interaction with the CC manager. According to the die or silent commit option, an injured txn is aborted only at the end of its execution [Robinson 1984; Ryu and Thomasian 1987]. No validation is required at the completion of the read phase when a txn is already injured; its access entries are deleted and the txn is aborted. Otherwise

(1) The relevant access entries are locked by an atomic action. The txn is simply delayed if there is a lock conflict until the required access entries are unlocked.

(2) The access entries of objects updated by the txn are used to injure txns that have accessed the same objects.

(3) The txn writes its log records, externalizes updated objects, and deletes access entries.

Aborted txns can be restarted right away, since the conflicting txns have committed and left the system; that is, no further conflicts will occur.

Both OCC and 2PL are inefficient when txns update database hot-spots. Locking has been extended with field calls in IMS FastPath to deal with aggregate variables constituting hot-spots [Gray and Reuter 1992]. The similarity between OCC and field calls considered in Gray and Reuter [1992] is repeated here in the context of an inventory control application [O'Neil 1987]. A txn first tests that enough items are available (e.g., $QOH \geq Request\_size$) to satisfy the *Request_size* for a certain order and again before txn commit. If the first (resp., second) test fails, then the txn follows an alternate path (resp., is aborted). This method is similar to OCC in that the object is not updated after the first test, but only after the second test is successful. Only one out of M txns accessing the QOH succeeds with OCC, whereas all M txns will succeed with the field-calls approach, as long as the current value of QOH exceeds the sum of *Request_sizes*.

4.2 Mechanisms Used by Two-Phase Processing Methods

In this section we describe mechanisms associated with txn scheduling methods that take advantage of access invariance.

(1) *Transaction phases.* A txn phase is an instance of its execution. The first execution phase of a txn may lead to txn commit, but even when it is unsuccessful (e.g., the txn fails its validation), it serves the role of priming the database buffer. We only distinguish among the execution in the first phase, which usually requires disk I/O, and further phases, which do not (provided access invariance prevails).

(2) *Running modes.* We consider three running modes:
   —Virtual Execution (VE). This execution mode is intended for determining the script of a txn (i.e., the set of objects accessed by the txn). Txn type and the input data to the txn provide some information about the future behavior of a txn, but this also depends on the state of the database. VE can be used to predict the data blocks accessed by a txn with some degree of certainty, by running a txn without invoking CC. The "pipelined" txn processing scheme in Reuter [1985] is based on VE in the first phase and serialized execution (SE), which may be considered a degenerate form of CC, in the second phase. One thread in a multithreaded experimental txn-processing system executes txns in SE mode, whereas other threads are used for execution in VE mode [Li and Naughton 1988].
   —Optimistic die and kill methods are applicable.
   —Locking. DL (dynamic locking) and SL (static locking) or lock preclaiming are relevant for the second phase (see Section 4.3), since they do not introduce any txn restarts (this is almost true in the case of DL), whereas restart-oriented locking methods, such as RP (running priority), are appropriate for the first phase.

An "aborted" txn releases its locks. If the txn is in its second phase it begins restart waiting, and first-phase txns continue their execution in the weaker VE or optimistic modes. In the latter case objects accessed/updated by "aborted" txns are copied into the txn's private workspace.
Switching from a weaker to a stronger mode, such as VE → OCC, VE → locking, or OCC → locking, is more difficult. Consider switching an uninjured txn running in optimistic mode to locking mode, as considered in Franaszek et al. [1991a] (see Section 4.3). Upgrading the access entries of a txn to locks may result in other txns being injured and txn updates being externalized at this point.

(3) *Locking priorities.* A two-phase processing method should specify how data conflicts are resolved among txns in the same and different phases.
   Locks are given a higher priority than access entries, since: (i) txns in optimistic mode are much more susceptible to aborts than txns with standard locking; and (ii) the optimistic mode is used in the first phase and is followed by the locking mode in further phases. We next consider lock conflicts.
   —*No locking priorities.* Lock requests are handled in FCFS order, regardless of the phase of the txn making the lock request.
   —*Nonpreemptive locking priorities.* Lock requests by second-phase txns are given a higher priority than first-phase txns.
   —*Preemptive locking priorities.* Lock requests by txns in the second or further phases are given preemptive priority with respect to locks held by first-phase txns. A txn that loses a lock in the first phase may continue running in virtual execution mode.

(4) *Transaction spawning.* In addition to simply blocking a txn when it encounters a lock conflict, it is possible to spawn a subtxn [Franaszek et al. 1992]. The subtxn that is provided with a private workspace runs in VE mode prefetching data required for the main txn's execution.

(5) *Checkpointing at the transaction level.* This type of checkpointing can be used to reduce the wasted processing caused by txn aborts. Checkpointing in OCC is discussed in Section 4.3.

(6) *CPU priorities.* Txns in phase two have preemptive CPU priority with respect to txns in phase one, because it is important to minimize the holding time for locks and access entries in this phase (see Section 4.3). Note that the execution time of txns in the first phase is dominated by disk I/O time and little is lost by having a lower CPU priority level in this phase. A txn in the first phase will run at an even lower priority level after it is conflicted; for example, the priority level of a txn running in optimistic die mode is lowered after a conflict.

Integrated CC and CPU scheduling [Franaszek et al. 1991b]. This paradigm is based on matching the number of executing txns with the availability of hardware resources in the system (e.g., CPU utilization). For example, txns in the system are prioritized based on whether they are runnable in the following categories: (1) standard locking; (2) RP, that is, txns not runnable by GW, but runnable with RP, for example, txn T_C in the WFG $T_C \rightarrow T_B \rightarrow T_A$; (3) the optimistic method, where all txns are runnable. Txns in category (1) run at priority level i (1 is the highest priority level). When the CPU is underutilized because there are no txns such as T_A to run, the scheduler will first run txns such as T_C. This seems to require the abort of T_B, but this is not required if a private workspace paradigm is adopted, which facilitates running in optimistic mode as well. If the CPU is still not fully utilized after running txns such as T_C, the scheduler will run all txns, including txns such as T_B. Conversely, more conservative running options are adopted when the CPU is fully utilized. Further investigations are required to ascertain the viability of this method.

4.3 Description and Performance of Two-Phase Processing Methods

In this section we describe txn scheduling methods utilizing the mechanisms in Section 4.2. VE and the optimistic die method are suitable for execution in the first phase, because they serve the purpose of prefetching data for the second execution phase. Lock or data contention is less of a problem in the second phase, since the mean number of txns executing in this phase, which determines the effective degree of txn concurrency, tends to be an order of magnitude smaller than the number of txns in the first phase. Hence the method used in the second phase has little effect on performance. However, to reduce the variability of txn response times, locking methods should be used in preference to OCC methods, since locking (with lock preclaiming and preemptive lock priorities for second-phase txns with respect to second-phase txns) disallows repeated txn aborts, whereas OCC allows restarts of second-phase txns by first-phase txns.

A list of viable two-phase processing methods is specified as "p1/p2-p2′ . . . ," where p1 is the CC method used in phase 1 and p2,p2′, . . . are the CC methods used in phase 2. In case a txn is aborted in phase 2, it is usually re-executed with the same CC method, but this is not always the case, for example, when access invariance is violated [Thomasian and Ryu 1990; Thomasian 1997b]. Representative two-phase processing methods are as follows:

(1) *VE/SE-DL-SL-optimistic kill.* SE runs txns serially in the second phase, whereas the other methods allow concurrent txn processing in this phase.

(2) *Optimistic die/kill.* The optimistic die method in the first phase serves the purpose of data prefetching, even though the txn has been conflicted, but its repeated use in further phases is not justifiable, since it no longer serves this purpose. Optimistic kill, on the other hand, minimizes wasted processing in latter phases.

(3) *Optimistic kill/kill.* The optimistic kill method is appropriate for the

first phase in the following cases: (i) txns are not access-invariant; (ii) the database is main-storage-resident or few disk accesses are required due to a high buffer hit ratio; and (iii) wasted processing cannot be tolerated due to limited processing capacity. The preceding comments about txn validation priorities also apply in this case.

(4) *Optimistic die/DL-SL*. These are examples of hybrid CC methods, which have the advantage with respect to optimistic die/kill that second-phase txns have priority with respect to first-phase txns; that is, lock requests conflicting with access entries result in a fatal injury to the corresponding first-phase txn, unless the txn making the lock request is aborted later to resolve a deadlock. Lock preclaiming in the second phase according to SL ensures that repeated aborts, which are inherent in OCC methods, are alleviated [Thomasian and Rahm 1990; Thomasian 1997b] (the advantage of this method is noted in Lynch et al. [1994]).

(5) *RPA_VE/DL-SL*. A txn in the first phase may be aborted due to the RPA (asymmetric RP) paradigm or due to lock preemption by a second-phase txn, since lock requests by txns in the second phase have a preemptive priority with respect to locks held by first-phase txns. A first-phase txn conflicted in this manner copies objects protected by its locks into a private workspace, releases its locks, and rather than being aborted, continues running in VE mode. A first-phase txn (running in RPA mode) is blocked when it has a lock-conflict with an active or blocked second-phase txn, because lock requests by second-phase txns have a higher priority than first-phase txns.

We use this context to elaborate on txn spawning at this point. There are two possibilities when a conflicted txn (say T) spawns a subtxn that starts running in VE mode.

—T is unblocked before the spawned subtxn completes its execution. There are two choices: abort the spawned subtxn or run the spawned subtxn to the end, so that no spawning will be required if the txn is blocked again. This is so provided access invariance prevails.

—The spawned subtxn completes while T is still blocked. In this case T is switched from the first to the second phase. In case T is blocked by a first-phase txn and provided that lock requests by second-phase txns have a preemptive priority with respect to first-phase txns, then the first-phase txn is aborted releasing its locks. If T is blocked by a second-phase txn, it will remain blocked until this txn completes.

Simulation results indicate that txn spawning does not improve performance with respect to RP-VE/DL for the range of modeling parameters considered [Franaszek et al. 1992]. Txn spawning has been adopted for real-time txn processing in Bestavros and Braoudakis [1995].

The branching txn paradigm [Burger and Thanisen 1992] in the context of multiversion locking provides for two instances of txn execution based on the old and new value of the locked variable. "Branch-restriction policies" are required, since otherwise the number of txns in the system grows exponentially, leading to thrashing due to resource exhaustion.

Polyvalues provide a similar capability for coping with failures in a distributed database environment through a bookkeeping tool described in Montgomery [1978]. With two pending txns T_1 and T_2 there are three polyvalues for the bank account balance ($B_{init} - W_1 - W_2$, T_1, T_2), ($B_{init} - W_1$, $T_1 \neg T_2$), and ($B_{init} - W_2$, $\neg T_1$, T_2), where B_{init} stands for initial bank balance, W for withdrawal, and T_i (resp., $\neg T_i$) indicates the commit (resp., abort) of txn T_i. Thus items remaining locked due to the failure of a commit coordinator can be accessed conditionally by other txns. Note that no branching is required by txns, which just require the balance to exceed a certain value. The escrow paradigm generalizes and formalizes polyvalues in the case of aggregate variables [O'Neil 1986].

The issue of accessing different versions of an object arises in the context of hybrid CC methods as well [Thomasian and Rahm 1990; Thomasian 1997b]. The alternatives are

(1) access committed data;
(2) access exclusively locked but uncommitted objects from the database buffer or the private workspace. Such objects may be modified again (i.e., do not necessarily reflect the version to be committed); and
(3) block access until all exclusive locks held on the object are released.

Simulation results show little difference between the performance attained by the first and third methods in the context of an optimistic die/SL hybrid method [Thomasian 1997b].

Performance comparison of two-phase processing methods. We summarize the simulation results in Franaszek et al. [1992], which considers a multiprocessor with a large database buffer. The load on the disks is assumed to be balanced and the number of disks is chosen such that all disks have a 20% utilization when the processors are 75% utilized. Disk utilization is based on a buffer hit ratio of 62.5% when all txns commit at the end of their first execution phase. The database buffer is large enough to ensure that txns running in their second phase can access data requested in the first phase from the buffer. Each txn requests 16 locks on the average, of which 25% are to hotspots. The comparison of CC methods is based on their ETC ($T(M)$, $M \geq 1$), that is, which CC method attains the maximum effective throughput.

(1) The VE/DL-SL method provides a lower bound to the ETC attainable by two-phase processing methods. There is no difference between the performance attained by using the DL and SL methods in the second phase, since there are few txns in this phase at any time and there is little lock contention. SL is preferable to DL in that it prevents txn aborts due to deadlocks. In a high-data-contention system with adequate hardware resources, VE/DL outperforms standard locking, which also follows from the numerical results obtained from the analysis in Thomasian and Ryu [1991].
(2) The optimistic die/kill method outperforms the optimistic kill/kill method in a system with adequate hardware resources and access invariance, but this is not so in a system with limited hardware resources [Franaszek et al. 1990, 1992]. This is because executing txns to the end ensures their successful re-execution if access invariance prevails, since txn execution time without disk I/O tends to be very short. The optimistic kill/kill method also exhibits the prefetching effect up to the point of txn abort, while wasting less processing.
(3) The optimistic die/kill method outperforms the RP_VE/SL-DL method in high-contention systems with adequate hardware resources and vice versa. The main reason is that the RP_VE running mode results in less wasted processing in the first phase than the OCC die policy.

One observation about two-phase processing methods is that their performance follows the same pattern at very high lock-contention levels, which is determined by the bottleneck hardware resource. This is because very few txns can commit successfully at the end of their first execution phase and hence both phases of almost all txns are executed. Two-phase processing methods may outperform the WDL method in high-data-contention systems with adequate hardware resources [Franaszek 1992]. However, this is not always so; for example, the WDL method outperforms the optimistic die/SL method in a system with a main storage database (no prefetching required) and a large number of processors [Franaszek 1992], but is outperformed by the optimistic kill method in this case.

An adaptive method to improve the performance of two-phase processing methods is described in Franaszek [1991a]. Txns are initially run in optimistic die/kill mode and once the bottleneck resource (e.g., the processors) becomes fully utilized (due to increased load) a more conservative method such as RPA is invoked. Further saturation of CPU utilization may lead to the adaptive variant of RPA described in Section 3.1. Simulation results in Franaszek [1991a] show that the improvement in performance due to adaptive methods can be significant.

Checkpointing in optimistic concurrency control. The effect of checkpoints or volatile savepoints [Gray and Reuter 1992]) by individual txns are investigated in the context of the optimistic kill method in Thomasian [1995b]. Checkpointing is appropriate for OCC since: (i) it solely uses aborts to resolve data conflicts and (ii) checkpointing is facilitated by the private workspace paradigm (i.e., the updates of aborted txns need not be undone). There is a tradeoff between checkpointing overhead and the saved processing due to partial rollbacks, which allows a txn to resume execution from the checkpoint preceding access to the data item to be released.

Numerical results obtained from an analytic solution for the optimistic kill method with checkpointing show that it is not effective if database objects are accessed uniformly, even if checkpointing overhead is low and a checkpoint is taken before each data access [Thomasian 1995b]. This is because the probability with which a data object is conflicted is proportional to the length of time since it was accessed, so that objects accessed at the beginning of txn execution are more susceptible to conflict than those accessed towards its completion. In the case where one checkpoint is to be taken, numerical results show that the mean response time and wasted processing are minimized if the checkpoint is taken after ⅓ of objects required by a txn are accessed (this number may not be known a priori), which implies that only ⅓ of the txn's processing is saved. Performance can be improved if accesses to hot-spots are deferred to the end of txn execution and a checkpoint is taken prior to these accesses, because objects held for a short duration of time are less vulnerable to conflicts and most of txn's processing is preserved if a data conflict occurs. This may be accomplished by rewriting the txn program, but this tends to be very costly.

Improving the performance of optimistic concurrency control. The distinction backward-oriented and forward-oriented validation in OCC is made in Haerder [1984], where the former corresponds to the validation method described in Section 4.1. In forward-oriented OCC a txn checks whether its write set conflicts with the read sets of txns in their read phase and the txn commits and "the write-sets are only propagated if they do not conflict with current read sets of all other active txns." Thus forward-oriented validation in OCC provides opportunities for performance improvement as follows.

(1) Defer txn commit when there is a conflict. A case when deferring txn validation results in a potentially improved performance is as follows. An object modified by the validated txn has been read by another txn. Deferring the introduction of the modified object into the database al-

lows the other txn to validate successfully. Consider T_1, which is ready to commit, having modified object A to A'. T_2, which is in its read phase, may have read A and used it to modify B to B'. Deferring T_1's commit to after T_2's commit results in the serialization order $T_1 \rightarrow T_2$.

(2) A validating txn that conflicts with other txns may commit suicide, although it has not been injured earlier, since this action is expected to improve overall performance (e.g., by reducing wasted processing).

Deferring the commit of a txn makes it vulnerable to conflicts with other txns. In addition, a txn's suicide to save another txn may be in vain, since the conflicting txn may be aborted after all. A dependency graph can be constructed among a set of txns to dynamically determine a commit ordering (different from the order in which txn executions are completed), which may improve performance with respect to basic OCC. Similarly to serialization graph testing [Bernstein et al. 1987], this is at the cost of added space and complexity. The validation scheme based on "intervals of timestamps" is intended to improve the success rate in a distributed database environment [Boksenbaum et al. 1987].

4.4 Performance Analysis of Optimistic Concurrency Control Methods

Analytic modeling methods for OCC and insights gained from previous studies are outlined in this section. The analyses of OCC methods are based on Menasce and Nakanishi [1982], Moenkeberg and Weikum [1985], and Ryu and Thomasian [1987].

An Analytic solution for optimistic concurrency control. The first paper dealing with the analysis of OCC considers the die method with static data access; that is, all objects required by a txn are accessed at the beginning of txn execution [Menasce and Nakanishi 1982]. We simplify the discussion by considering a closed system with M txns.

The probability of a data conflict in a database with size D when the committing (resp., conflicting) txn is updating m (resp., accessing n) data objects, which are accessed with uniform probabilities, is given by

$$\phi(n, m) = 1 - \binom{D-n}{m} \Big/ \binom{D}{m} \simeq 1 - (1 - n/D)^m \simeq nm/D.$$

In the case of fixed-size txns with size k, where all k accessed objects are updated, $\psi = \phi(k, k) \simeq k^2/D$ [Franaszek et al. 1992].

A key observation from the preceding equation (and the simple analysis in Franaszek et al. [1992]) is that the probability of txn abort increases with the square of its size, which is referred to as the quadratic effect in Franaszek et al. [1992]. Thus when txn steps have equal processing times, the probability of conflict with an optimistic die (resp., kill) method increases proportionally to k^2/D (resp., $k^2/(2D)$). It is observed from numerical results in Ryu and Thomasian [1987] that in a system with variable txn sizes the wasted processing is dominated by the largest txn sizes.

The system is represented by a Markov-chain model whose states (S_j) specify the number of txns (j) that can complete successfully. The transition rates of the Markov chain are obtained by solving the QNM of the underlying computer system to obtain the STC $t(M)$, $M \geq 1$. The state transitions are affected by the probability that a committing txn at S_j conflicts with l txns from the remaining $j - 1$ txns, which is given by

$$\binom{j-1}{l} \psi^l (1 - \psi)^{j-1-l}.$$

The analysis in Menasce and Nakanishi [1982] underestimates the mean response times attained by OCC [Morris and Wong 1985] because OCC favors the successful validation of txns with shorter processing times. The processing times of restarted txns, which are assumed to be exponentially distributed in the Markov-chain analysis, are resampled when a txn is restarted. The mean execution times of txns completed by the system tends to be less than the intended average. It can be argued, however, that the processing times of txn phases vary from one execution to another and that txns with shorter execution times have a higher probability of success.

We next outline the analysis in Morris and Wong [1985]. Txn processing times are postulated to be exponentially distributed (with mean $1/\mu$) and the effect of hardware resource contention is specified by the processing rate $s(M)$ such that txns proceed with rate $\mu s(M)$. Let u denote the system efficiency or the fraction of time the system spends doing useful work. Then the txn completion rate is $T(M) = u\mu s(M)$. A key assumption used in the analysis is that running txns observe the commits of other txns as a Poisson process with rate $\lambda = (1 - 1/M)u\mu s(M)$. The rate at which a txn is conflicted is then $\gamma = \lambda\psi$. The probability of a data conflict for a txn with processing time x is $q = 1 - e^{-\gamma x M/s(M)}$. Provided that the execution time of a txn is not resampled (it remains equal to x), the number of its executions in the system follows a geometric distribution $P_j = q(1 - q)^{j-1}$, $j \geq 1$ with a mean $\bar{J}(x) = 1/q = e^{\gamma x M/s(M)}$. The analysis in Menasce and Nakanishi [1992] assumes that the number of txn executions ($\tilde{J}$) is independent of txn execution time ($\tilde{X}$) and hence $E[\tilde{J}\,\tilde{X}] = \bar{J}\,\bar{X}$.

The system efficiency can be specified as the ratio of the mean execution time of a txn and its residence time in the system:

$$u = \frac{E[xM/s(M)]}{E[xMe^{\gamma x M/s(M)}/s(M)]}$$

$$= \frac{\int_0^\infty xe^{-\mu x}dx}{\int_0^\infty xe^{-(\mu-\gamma M/s(M))x}dx}$$

$$= [1 - (M - 1)\psi u)]^2.$$

$\gamma M/s(M) < \mu$ is required for the denominator to converge. Solving the quadratic equation yields one acceptable root: $u = (1 + 2a - \sqrt{1 + 4a})/4a^2 \simeq 1 - 2a$, where $a = (M - 1)\psi$ is small, such that higher terms in expanding $(1 + 4a)^{1/2}$ can be ignored.

This study similarly to Menasce and Nakanishi [1982] considers the relative performance of SL and the optimistic die method. SL (with the FCFS with skip option; see Section 2.3) outperforms the optimistic die method, whereas the more efficient optimistic kill method (with static data access) has a performance indistinguishable from SL in a system with infinite resources [Thomasian and Ryu 1986], where the wasted processing time due to txn restarts does not affect the processing time of other txns.

These analyses are extended in several directions in Ryu and Thomasian [1987]. Numerical results show that the txn-processing-time distribution affects the overall performance. The analysis of the kill method with exponential processing times yields $u = (1 + a)^{-1} \simeq 1 - a$, which indicates that the die method is twice as inefficient as the kill method. The analysis of OCC with dynamic (on demand) object accesses takes into account the fact that the conflict rate varies according to the number of data objects that have been accessed. The analysis of a system with multiple txn classes, where class is determined by txn size, considers each class separately to ensure that the fraction of txns in the stream of completed txns is the same as in the arrival stream (see Section 3.3). The analysis can be extended to take into account the variability in txn processing times across executions (e.g., due to the prefetching effect). Also, different methods can be modeled for different phases (e.g., optimistic die in the first phase and optimistic kill or locking in the second phase).

5. CONCLUSIONS

The performance of an abstract standard locking model is analyzed in this article to provide an understanding of factors leading to its performance degradation. Restart-oriented locking methods relieve lock contention by selectively aborting txns, and two-phase processing methods reduce the data contention level in systems with access invariance by shortening the holding time of locks or access entries by prefetching the data required for txn execution in the second phase, if it is required. We summarize the conclusions of previous simulation and analytic studies regarding the relative performance of CC methods and survey methods applicable to the analysis of standard locking, restart-oriented locking methods, and OCC.

Conclusions based on simulation and analytic studies regarding the relative performance of CC methods can be summarized as follows.

(1) Standard locking is desirable in low-lock-contention environments, since it introduces a minimal amount of extra processing by aborting txns only to resolve deadlocks. The level of lock contention is expected to be low in relational DBMSs fine-tuned for txn processing applications [Mohan et al. 1992a].

(2) Given that there are no robust load-control methods to prevent thrashing, especially in environments with varying workloads, restart-oriented methods such as the WDL method with restart-waiting provide an easy-to-implement solution to this problem. In fact, simulation results show that WDL attains a much higher txn throughput than standard locking and most other methods in a high-lock-contention environment.

(3) Two-phase processing methods are beneficial in systems with longer txn response times due to disk I/O or remote accesses, resulting in higher degrees of txn concurrency and hence higher data contention. Simulation results show that two-phase methods result in significantly improved performance with respect to standard locking, as well as restart-oriented locking methods such as WDL, provided adequate hardware resources are available. Specialized software can be used to reduce the overhead of executing txns in the second phase [Reuter 1985]. In addition, provisions need to be made to reduce the overhead of undoing the updates of aborted txns, which also applies to restart-oriented locking methods.

Key insights derived from analytic and simulation studies are summarized as follows.

(1) The single metric α (defined in Section 2.2) uniquely determines the lock-contention level in a standard locking system with the homogeneous txn access model and identical per-step processing times. The system thrashes beyond the critical value $\alpha^* = 0.226$, which is just beyond the point where the maximum throughput in a system with infinite resources is attained. The mean number of active txns ($\bar{M}_a$) is maximized at the point where 30% of txns are blocked ($\beta \simeq 0.3$), which holds for various txn size distributions and txns with different per-step processing times [Thomasian 1993].
In a system with finite resources the maximum throughput is determined by the bottleneck resource (see Section 1.3), which is usually at a relatively low txn-concurrency level resulting in a low lock-contention level.

(2) Factors leading to thrashing are (i) a sudden increase in the number of

activated txns; (ii) although the limit on the degree of txn concurrency to prevent thrashing is not exceeded, an undesirable composition of txns is activated (e.g., all activated txns are long); and (iii) there is a temporary increase in the number of accesses to hot-spots.

(3) For txns of size k and identical per-step processing times, the probability of lock conflict (P_c) (resp., the probability of deadlock per txn (P_D)), is proportional to k^2 (resp., k^4) [Gray and Reuter 1992]. For txns with a variable number of lock requests, P_c depends on the second moment of requested locks (see Eq. (2.8) for the mean number of locks held per txn) and P_D depends on the third moment of txn size [Thomasian and Ryu 1991]. Thus the variability of txn size has a major effect on the lock-contention level.

(4) The mean waiting time for a lock held by an active txn (W_1) is one third of txn's mean residence time, provided that txns have a fixed size and locks are requested uniformly over its residence time (see Eq. (3.5)). The mean waiting time per lock request (W) is a weighted sum of the probability of being blocked at level i ($P_b(i)$) and the associated mean waiting time (W_i). It follows from Eq. (2.10) that $W \simeq 1.8W_1$ at the point where the mean number of active txns ($\bar{M}_a$) is maximized at $\beta = 0.3$.

In the case of variable-size txns with identical steps, W_1 is proportional to the third moment of the number of requested locks. It follows that the variability of txn size has a major effect on the performance degradation due to lock contention in this case.

(5) When the per-step processing times are different, in addition to α, the fraction of lock conflicts with blocked txns (ρ) is a useful measure of lock contention in standard locking. Experimentation shows that as the level of lock contention is varied $0.2 \leq \rho \leq 0.3$ [Thomasian 1996b], which is consistent with the conflict-ratio metric equal to $(1 - \rho)^{-1}$ [Moenkeberg and Weikum 1992]. Similarly to txns with identical per-step processing times, $\bar{M}_a$ in this case is also maximized at $\beta \simeq 0.3$.

(6) For the heterogeneous database-access model ρ was observed to vary over a wider range than for the homogeneous database-access model. Although concurring with experimental results in Weikum et al. [1994], the wider range of values for ρ makes it less useful for load control.

(7) Restart-oriented locking methods reduce the level of lock contention at the cost of additional processing. In a lock-contention-bound system significant increases in the maximum throughput attainable by the system are possible. The WDL [Franaszek et al. 1992] and MWDL [Thomasian 1992; 1997] methods outperform RPS; because they cause less wasted processing than RPS. This is accomplished through heuristics that take into account txn progress. The RPA method can outperform both WDL and MWDL methods in a system with limited hardware resources because it results in less wasted processing than both of the preceding methods and RPS.

Adaptive methods that vary the wait depth and breadth to maximize CPU utilization, without saturating it, tend to improve system performance.

(8) Optimistic CC methods are susceptible to abort according to a quadratic effect; that is, the probability of an unsuccessful validation increases as the square of txn size [Franaszek et al. 1992]. Numerical results based on the performance analysis in Ryu and Thomasian [1987] show that in a system with variable txn sizes most of the wasted processing is due to the larger txn sizes.

(9) Two-phase txn-processing methods reduce the effective level of txn concurrency to attain higher txn throughputs. Thus even if the first execution phase of a txn is unsuccessful, it has the beneficial effect of prefetching the data from disk, which can be used by the second execution phase provided access invariance prevails.

There has been little work on analyzing the performance of txn processing systems taking into account lock-contention effects. This is due to the difficulty of characterizing txn processing systems from the lock-contention viewpoint, rather than the inability to develop appropriate methods for performance evaluation (a simulation study can be used when analysis is not possible). The locking analysis in Thomasian [1996b] (see Section 2.5), with appropriate extensions (e.g., to take into account specialized locking methods for index structures), might lend itself to "answering engineering questions" [Tay 1990], for example, predicting the effect of lock contention on the performance of a txn-processing system. Analytic results need be validated against measurement results rather than just simulations.

Statistics on lock-contention levels are provided by most DBMSs, but the measurement data are not sufficiently detailed and cannot be easily correlated with other events in the system. Furthermore, additional information is required about the database, txns processed by the system, and txn scheduling to synthesize a realistic lock-contention model. Some of this information is proprietary in nature and not available externally.

Some additional topics related to CC are as follows.

CC in "advanced database applications" or "unconventional txn management" is an area of current interest [Barghouti and Kaiser 1991; Elmagarmid 1992; Ramamithram and Chrisanthis 1996]. Alternative txn models such as sagas, altruistic locking, cooperating txns, and txn chopping, which were discussed in Section 2.6, are a step in this direction. Some of the associated problems have to do more with scheduling and coordinating related activities, rather than CC, such as in the case of workflow models [Khoshafian and Buckiewicz 1995].

Real-time txns or databases are another area of current research. Various CC methods have been evaluated to determine their suitability for real-time txn processing and many new CC methods have been proposed [Ramamithram 1993]. These include methods based on access invariance, because it provides an opportunity to preanalyze the objects required by the txn for its second phase execution [O'Neil et al. 1995] and hybrid methods (see Section 4.3) [Huang et al. 1991]. Txn spawning for improving performance in real-time systems has been considered in Bestavros and Braoudakis [1995].

Active databases provide timely responses to time-critical events and this is accomplished by providing event-condition-action rules to be specified for the DBMS. A performance evaluation of active databases that takes into account locking effects and distinguishes between external and rule-management tasks is reported in Carey et al. [1991].

CC in distributed databases is covered in an early tutorial [Bernstein and Goodman 1981] and several texts [Date 1983; Ceri and Pelagatti 1984; Bernstein et al. 1987; Cellary et al. 1988]. In fact, some early CC methods, such as TSO, WW, and WD methods, were intended for distributed databases. Standard locking with two-phase commit is the main approach used for CC and to ensure atomicity. The number of messages required for txn execution is quite important in this case [Ceri and Pelagatti 1984]. This includes the number of messages required for deadlock detection and two-phase commit. Methods to reduce the number of messages for commit processing are discussed in Samaras et al. [1995].

Numerous CC methods have been considered for distributed databases. The tradeoff between the number of messages and the information available to minimize the wasted processing due to txn aborts is explored for several variations of the distributed WDL method in Franaszek et al. [1993]. A method based on access invariance that uses OCC in the first phase and lock preclaiming in the second phase is discussed in Thomasian and Rahm [1990] and Thomasian [1998] (these papers also discuss earlier works in OCC in distributed databases). A comprehensive simulation study of several CC methods for distributed databases is reported in Carey and Livny [1991]. An example of an approach based on allocating fractions of aggregate values to the nodes of a distributed system is described in Thomasian [1994] (this paper also surveys related work on this topic).

In addition to multiprocessors, shared-nothing and shared-disk systems are of current interest for txn processing. In shared-nothing systems the data are partitioned to balance the load (e.g., by using hashing on primary keys in a relational database). From the viewpoint of CC these systems behave as distributed databases, although some optimizations are possible, for example, broadcast capability to all processors for two-phase commit. The simulation of a preliminary version of the TPC-C benchmark on a shared-nothing system appears in Jenq et al. [1989].

In shared-disk or data-sharing systems multiple computers have access to a set of shared disks. Txns are routed to computers to achieve a balanced load and to attain a higher database buffer hit ratio [Rahm 1993]. In addition to concurrency control, there is the issue of coherency control for the contents of database buffers across systems [Rahm 1993]. Similar problems arise in client-server systems, which are addressed in a simulation study reported in Carey et al. [1991].

Multidatabase txn management deals with distributed heterogeneous DBMSs, possibly with different CC and txn recovery methods. Local txns are run under the control of a local DBMS, and a multidatabase system is provided to run global txns. The multidatabase system is to accomplish its task with minimal modifications to the operation of the system [Breitbart et al. 1992]. As noted in Breitbart et al. [1992], most research to date has been concerned with how to run txns in a heterogeneous environment, with no attention paid to performance issues: for example, how much more expensive will it be to run txns when each box runs a different CC protocol? A hierarchical architecture for multidatabase systems is described in Mehrotra et al. [1997] and techniques for CC in such systems are developed.

Glossary

CC	concurrency control
CPU	central processing unit
CWA	cautious waiting—asymmetric [Hsu and Zhang 1992]
CWS	cautious waiting—symmetric [Thomasian 1997a]
DBMS	data base management system
DL	dynamic locking
EDSP	effective database size paradigm
ETC	effective throughput characteristic
FCFS	first-come, first-served
GW	general waiting method (i.e., standard locking)
MWDL	modified wait-depth-limited method [Thomasian 1992]
NW	no waiting
OCC	optimistic CC [Kung and Robinson 1981]
QNM	queueing network model
RPA	running priority—asymmetric [Franaszek and Robinson 1985]
RPS	running priority—symmetric [Franaszek et al. 1992]
SE	serialized execution

SL	static locking
STC	system-throughput characteristic
TSO	time-stamp ordering method
Txn	Transaction
VLDB	Very large data bases
WDL	wait-depth limited [Franaszek et al. 1992]
WD	wait-die [Rosenkrantz et al. 1978]
WFG	waits-for graph
WW	wound-wait [Rosenkrantz et al. 1978]
2PL	two-phase locking

Appendix: Notation

α	Mean number of lock conflicts per txn times the normalized waiting time for single-level blocking ($\alpha = K_1 P_c A$ with $A = W_1/R(M)$).
β	Fraction of txns in the blocked state ($\beta = K_1 P_c W/R(M) = \bar{M}_b/M$).
C_k	Txns in class k that request k locks.
D	Number of data items and associated locks in the database.
f_k	Fraction of txns in class k (C_k) that also request k locks.
K	Largest txn size or maximum number of locks requested per txn.
K_i	ith moment of the number of requested locks.
$\bar{L}_a(\bar{L}_b)$	Mean number of locks held by a txn while it is active (blocked) ($\bar{L} = \bar{L}_a + \bar{L}_b$).
$\bar{L}_k$	Mean number of locks held by txns in C_k.
M	Number of txns in the closed system.
$\bar{M}_a(\bar{M}_b)$	Mean number of active (blocked) txns in the system ($\bar{M}_a = (1 - \beta)M$ and $\bar{M}_b = \beta M$).
$\hat{M}$	Multiprogramming level that maximizes the number of active txns in the system ($\bar{M}_a$) and potentially system throughput.
n_c	Mean number of lock conflicts per txn ($n_c = K_1 P_c$).
P_c	Probability of lock conflict per lock request ($P_c \simeq (M - 1)\bar{L}/D$).
$P_D(i)$	Probability that a txn encounters a deadlock of cycle length i.
$r_k(M)$	Mean response time for a txn in C_k as determined by hardware-resource contention only.
$r(M)$	Mean response time over all txn classes as determined by hardware-resource contention only ($r(M) = \sum_{k=1}^{K} r_k(M) f_k$).
$R_k(M)$	Mean response time for txns in C_k ($R_k(M) = (k + 1)s(\bar{M}_a) + kP_c W$).
$R(M)$	Mean response time over all txn classes ($R(M) = \sum_{k=1}^{K} R_k(M) f_k$).
ρ	Conflict ratio, which is the fraction of lock conflicts that are with blocked txns.
$s(\bar{M}_a)$	Mean processing time of a txn step with $\bar{M}_a$ active txns in the system.
$t(M)$	System throughput with M txns as determined by hardware-resource contention. $t(M)$, $M \geq 1$ is referred to as the throughput characteristic.
$T(M)$	Txn throughput with M txns in the system as affected by hardware-resource and lock contention. $T(M)$, $M \geq 1$ is referred to as the effective throughput characteristic of the system.
W_1	Mean txn waiting time due to a lock conflict with an active txn.
W	Mean txn waiting time due to a lock conflict.

REFERENCES

AGRAWAL, D., EL ABBADI, A., AND LANG, A. E. 1994. The performance of protocols based on locks with ordered sharing. *IEEE Trans. Knowl. Data Eng.* 6, 5 (Oct.), 805–818.

AGRAWAL, R., CAREY, M. J., AND LIVNY, M. 1987a. Concurrency control performance modeling: Alternatives and implications. *ACM Trans. Database Syst. 12,* 4 (Dec.), 609–654.

AGRAWAL, R., CAREY, M. J. AND MCVOY, L. W. 1987b. The performance of alternative strategies for dealing with deadlocks in database management systems. *IEEE Trans. Softw. Eng. 13,* 12 (Dec.), 1348–1363.

AMMANN, P., JAJODIA, S., AND RAY, I. 1997. Applying formal methods to semantic-based decomposition of transactions. *ACM Trans. Database Syst. 22,* 2 (June), 215–254.

BADRINATH, B. R. AND RAMAMITHRAM, K. 1992.

Semantics-based concurrency control: Beyond commutativity. *ACM Trans. Database Syst. 17,* 1 (March), 163–199.

BARGHOUTI, N. S. AND KAISER, G. E. 1991. Concurrency control in advanced database applications. *ACM Comput. Surv. 23,* 3 (Sept.), 269–317.

BAYER, R. 1986. Consistency of transactions and random batch. *ACM Trans. Database Syst. 11,* 4 (Dec.), 397–404.

BERNSTEIN, P. A. AND GOODMAN, N. 1981. Concurrency control in distributed database systems. *ACM Comput. Surv. 13,* 2 (June), 185–221.

BERNSTEIN, P., HADZILACOS, V., AND GOODMAN, N. 1987. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley, Reading, MA.

BESTAVROS, A. AND BRAOUDAKIS, S. 1995. Value-cognizant speculative concurrency control. *Proceedings of the 21st International Conference on Very Large Data Bases* (Zurich, Switzerland, Sept.) 122–133.

BOBER, P. M. AND CAREY, M. J. 1992. On mixing queries and transactions via multiversion locking. *Proceedings of the Eighth International Conference on Data Engineering* (Tempe, AZ, Feb.), 535–545.

BOKSENBAUM, C., CART, M., FERRIE, J., AND PONS, J. F. 1987. Concurrent certification by interval of timestamps in distributed database systems. *IEEE Trans. Softw. Eng. 13,* 4 (April), 409–419.

BREITBART, Y., GARCIA-MOLINA, H., AND SILBERSCHATZ, A. 1992. Overview of multidatabase transaction management. *Int. J. Very Large Data Bases 1,* 181–239.

BURGER, A. AND THANISCH, P. 1994. Branching transactions: A transaction model for parallel database systems. In *Directions in Databases: Proceedings of the Twelfth British National Conference on Databases, BNCOD 12* (Guildford, UK, July), D. S. Bowers, Ed., 121–136.

CAREY, M. J. AND LIVNY, M. 1991. Conflict detection tradeoffs for replicated data. *ACM Trans. Database Syst. 16,* 4 (Dec.), 703–746.

CAREY, M. J., JAUHARI, R., AND LIVNY, M. 1991. On transaction boundaries in active databases: A performance perspective. *IEEE Trans. Knowl. Data Eng. 3,* 3 (Sept.), 320–336.

CAREY, M. J., FRANKLIN, M. J., LIVNY, M., AND SHEKITA, E. J. 1991. Data caching tradeoffs in client server DBMS architectures. In *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data* (Denver, CO, May) 357–366.

CELLARY, W., GELENBE, E., AND MORZY, T. 1988. *Concurrency Control in Distributed Databases.* North-Holland.

CERI, S. AND PELAGATTI, G. 1984. *Distributed Databases—Principles and Systems.* McGraw-Hill, New York.

CHESNAIS, A., GELENBE, E., AND MITRANI, I. 1983. On the modeling of parallel access to shared data. *Commun. ACM 26,* 3 (March), 198–202.

DATE, C. J. 1983. *An Introduction to Database Systems, Vol. II.* Addison-Wesley, Reading, MA.

ELMAGARMID, A. K. 1992. *Database Transaction Models for Advanced Applications.* Morgan-Kaufmann, San Mateo, CA.

FARRAG, A. A. AND OZSU, M. T. 1989. Using semantic knowledge of transactions to increase concurrency. *ACM Trans. Database Syst. 14,* 4 (Dec.), 503–525.

FRANASZEK, P. AND ROBINSON, J. T. 1985. Limitations of concurrency in transaction processing. *ACM Trans. Database Syst. 10,* 1 (March), 1–28.

FRANASZEK, P. A., HARITSA, J. R., ROBINSON, J. T., AND THOMASIAN, A. 1993. Distributed concurrency control based on limited wait depth. *IEEE Trans. Parallel Distrib. Syst. 4,* 11 (Nov.), 1246–1264.

FRANASZEK, P., ROBINSON, J. T., AND THOMASIAN, A. 1990. Access invariance and its use in high contention environments. In *Proceedings of the Sixth International Conference on Data Engineering* (Los Angeles, Feb.), 47–55.

FRANASZEK, P. A., ROBINSON, J. T., AND THOMASIAN, A. 1991a. Adaptive concurrency control scheme for transaction processing. *IBM Tech. Disclosure Bull. 33,* 9 (Feb.), 29–30.

FRANASZEK, P. A., ROBINSON, J. T., AND THOMASIAN, A. 1991b. Integrated concurrency control/CPU scheduling. *IBM Tech. Disclosure Bull. 33,* 9 (Feb.), 37–40.

FRANASZEK, P., ROBINSON, J. T., AND THOMASIAN, A. 1992. Concurrency control for high contention environments. *ACM Trans. Database Syst. 17,* 2 (June), 304–345.

GALLER, B. I. AND BOS, L. 1983. A model of transaction blocking in databases. *Perform. Eval. 3,* 95–122.

GARCIA-MOLINA, H. 1983. Using semantic knowledge for transaction processing in a distributed database. *ACM Trans. Database Syst. 8,* 2 (June), 186–213.

GARCIA-MOLINA, H. AND SALEM, K. 1987. Sagas. In *Proceedings of the 1987 SIGMOD Conference on Management of Data* (San Francisco, May), 249–259.

GRAY, J. N., Ed. 1993. *The Benchmark Handbook for Transaction Processing Systems,* 2nd ed. Morgan-Kaufmann, San Mateo, CA.

GRAY, J. N. AND REUTER, A. 1992. *Transaction Processing: Concepts and Facilities.* Morgan-Kaufmann, San Mateo, CA.

HAERDER, T. 1984. Observations on optimistic concurrency control schemes. *Inf. Syst. 9,* 2, 111–120.

HAERDER, T. AND ROTHERMEL, K. 1993. Concurrency control issues in nested transactions. *Int. J. Very Large Data Bases 2,* 39–74.

HSU, M. AND ZHANG, B. 1992. Performance evaluation of cautious waiting. *ACM Trans. Database Syst. 17,* 3 (Sept.), 477–512.

HSU, M. AND ZHANG, B. 1995. Modeling performance impact of hot spots. In *Performance of Concurrency Control Mechanisms in Centralized Database Systems,* V. Kumar, Ed. Prentice-Hall, Englewood Cliffs, NJ, Chapter 7, 148–164.

HUANG, J., STANKOVIC, J. A., RAMAMITHRAM, K., AND TOWSLEY, D. 1991. Experimental evaluation of real-time optimistic concurrency control schemes. In *Proceedings of the Seventeenth International Conference on Very Large Data Bases* (Barcelona, Sept.), 35–46.

IRANI, K. B. AND LIN, H. L. 1979. Queueing network models for concurrent transaction processing in database systems. In *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data* (Boston, June), 134–142.

JAGADISH, H. V. AND SHMUELLI, O. 1992. A proclamation based model for cooperating transactions. In *Proceedings of the Eighteenth International Conference on Very Large Data Bases* (Vancouver, Canada, August), 265–276.

JENQ, B. C., TWICHELL, B. C., AND KELLER, T. W. 1989. Locking performance in a shared nothing parallel database machine. *IEEE Trans. Knowl. Data Eng. 1,* 4 (Dec.), 530–543.

JOHNSON, T. AND SHASHA, D. 1993. The performance of concurrent B-tree algorithms. *ACM Trans. Database Syst. 18,* 1 (March), 51–101.

KHOSHAFIAN, S. AND BUCKIEWICZ, M. 1995. *Introduction to Groupware, Workflow, and Workgroup Computing.* John Wiley, New York.

KLEINROCK, L. 1975. *Queueing Systems, Vol. I: Theory.* John Wiley, New York.

KORTH, H. F. AND SPEEGLE, G. 1994. Formal aspects of concurrency control in long duration transaction systems using the NT/PV model. *ACM Trans. Database Syst. 19,* 3 (Sept.), 492–535.

KRISHNAKUMAR, N. AND BERNSTEIN, A. J. 1994. Bounded ignorance: A technique for increasing concurrency in a replicated system. *ACM Trans. Database Syst. 19,* 4 (Dec.), 586–625.

KUMAR, V., Ed. 1995. *Performance of Concurrency Control Mechanisms in Centralized Database Systems.* Prentice-Hall, Upper Saddle River, NJ.

KUNG, H. T. AND ROBINSON, J. T. 1981. On optimistic concurrency control methods. *ACM Trans. Database Syst. 6,* 2 (June), 213–226.

LANGER, A. M. AND SHUM, A. W. 1982. The distribution of granule accesses made by database transactions. *Commun. ACM 25,* 11 (Nov.), 831–832.

LAZOWSKA, E. D., ZAHORJAN, J., GRAHAM, G. S., AND SEVCIK, K. C. 1984. *Quantitative System Performance: Computer System Analysis Using Queueing Network Models.* Prentice-Hall, Upper Saddle River, NJ.

LI, K. AND NAUGHTON, J. F. 1988. Multiprocessor main memory transaction processing. In *Proceedings of the International Symposium on Databases in Parallel and Distributed Systems* (Austin, TX, Dec.), 177–187.

LYNCH, N., MERRITT, M., WEIHL, W., AND FEKETE, A. 1994. *Atomic Transactions.* Morgan-Kaufmann, San Mateo, CA.

MASSEY, W. 1986. A probabilistic analysis of database systems. In *Proceedings of Performance 86 and ACM SIGMETRICS 1986 Joint Conference* (Raleigh, NC, May), 141–146.

MEHROTRA, S., KORTH, H. F., AND SILBERSCHATZ, A. 1997. Concurrency control in hierarchical multidatabase systems. *Int. J. Very Large Data Bases 6,* 157–172.

MENASCE, D. A. AND NAKANISHI, T. 1982. Optimistic versus pessimistic concurrency control mechanisms in database management systems. *Inf. Syst. 7,* 1, 13–27.

MOENKEBERG, A. AND WEIKUM, G. 1992. Performance evaluation of an adaptive and robust load control method for the avoidance of data contention thrashing. In *Proceedings of the Eighteenth International Conference on Very Large Data Bases* (Vancouver, Canada, Aug.), 432–443.

MOHAN, C. 1992. Less optimism about optimistic concurrency control. In *Proceedings of the Second International Workshop on Research Issues on Data Engineering* (Tempe, AZ, Feb.), 199–204.

MOHAN, C. AND LEVINE, F. 1992. ARIES/IM: An efficient and high concurrency index management method using write-ahead logging. In *Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data* (San Diego, June), 371–380.

MOHAN, C., HADERLE, D., LINDSAY, B., PIRAHESH, H., AND SCHWARTZ, P. 1992a. ARIES: A transaction recovery method supporting fine granularity locking and partial rollbacks using write-ahead locking. *ACM Trans. Database Syst. 17,* 1 (March), 94–162.

MOHAN, C., PIRAHESH, H., AND LORIE, R. 1992b. Efficient and flexible methods for transient versioning of records to avoid locking by read-only transactions. In *Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data* (San Diego, June), 124–133.

MONTGOMERY, W. A. 1978. Robust concurrency control for a distributed information system. MIT-LCS-TR-207, MIT, Lab. for Computer Science, Cambridge, MA. (Dec.).

MORRIS, R. J. T. AND WONG, W. S. 1985. Performance analysis of locking and optimistic concurrency control algorithms. *Perform. Eval. 5,* 2, 105–118.

MOSS, J. E. B. 1985. *Nested Transactions: An Approach to Reliable Distributed Computing.* MIT Press, Cambridge, MA.

O'NEIL, P. E. 1986. The escrow transaction method. *ACM Trans. Database Syst. 11,* 4 (Dec.), 405–430.

O'NEIL, P. E., RAMAMITHRAM, K., AND PU, C. 1995. A two-phase approach to predictability scheduling real-time transactions. In *Performance of Concurrency Control Methods in Centralized Database Systems* V. Kumar, Ed., Prentice-Hall, Englewood Cliffs, NJ, 494–522.

OZSU, M. T. 1994. Transaction models and transaction management in object-oriented database management systems. In *Advances in Object-Oriented Database Systems.* A. Dogac, M. T. Ozsu, A. Biliris, and T. Sellis, Eds. Springer-Verlag, New York, 147–184.

PEINL, P., REUTER, A., AND SAMMER, H. 1988. High contention in a stock trading database: A case study. In *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data* (Chicago, June), 260–268.

POTIER, D. AND LEBLANC, P. 1980. Analysis of locking policies in database management systems. *Commun. ACM 23,* 10 (Oct.), 584–593.

PU, C. AND LEFF, A. 1991. Replica control in distributed databases: An asynchronous approach. In *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data* (Denver, CO, May), 377–386.

RAHM, E. 1993. Empirical performance evaluation of concurrency and coherency control protocols for database sharing systems. *ACM Trans. Database Syst. 18,* 2 (June), 333–377.

RAMAMITHRAM, K. 1993. Real-time databases. *Distrib. Parallel Databases 1,* 199–226.

RAMAMITHRAM, K. AND CHRISANTHIS, P. K. 1996. *Advances in Concurrency Control and Transaction Processing.* IEEE Computer Society Press, Los Alamitos, CA.

REUTER, A. 1985. The transaction pipeline processor. In *International Workshop on High Performance Transaction Systems* (Pacific Grove, CA, Sept.).

ROBINSON, J. T. 1982a. Design of concurrency controls for transaction processing systems. Tech. Rep. CMU-CS-82-114, Ph.D. Thesis, Computer Science Dept., Carnegie-Mellon University, Pittsburgh, PA.

ROBINSON, J. T. 1982b. Experiments with transaction processing on a multiprocessor. IBM Res. Rep. RC 9725, Yorktown Heights, NY, Dec.

ROBINSON, J. T. 1984. Separating policy from correctness in concurrency control design. *Softw. Pract. Exper. 14,* 9 (Sept.), 827–844.

ROSENKRANTZ, D. J., STEARNS, R. E., AND LEWIS, P. M., II. 1978. System level concurrency control for distributed database systems. *ACM Trans. Database Syst. 3,* 2 (June), 178–198.

RYU, I. K. AND THOMASIAN, A. 1987. Performance evaluation of centralized databases with optimistic concurrency control. *Perform. Eval. 7,* 3, 195–211.

RYU, I. K. AND THOMASIAN, A. 1988. Performance analysis of centralized databases with static locking. Unpublished rep., IBM T. J. Watson Research Center, Hawthorne, NY.

RYU, I. K. AND THOMASIAN, A. 1990a. Analysis of database performance with dynamic locking. *J. ACM 37,* 3 (July), 491–523.

RYU, I. K. AND THOMASIAN, A. 1990b. Performance analysis of dynamic locking with the no-waiting policy. *IEEE Trans. Softw. Eng. 16,* 7 (July), 684–698.

SALEM, K., GARCIA-MOLINA, H., AND SHANDS, J. 1994. Altruistic locking. *ACM Trans. Database Syst. 19,* 1 (March), 117–165.

SAMARAS, G., BRITTON, K., CITRON, A., AND MOHAN, C. 1995. Two-phase commit optimizations in a commercial distributed environment. *Distrib. Parallel Databases,* 325–360.

SHASHA, D., LLIRBAT, F., SIMON, E., AND VALDURIEZ, P. 1995. Transaction chopping: Algorithms and performance studies. *ACM Trans. Database Syst. 20,* 3 (Sept.), 325–363.

SINGHAL, M. 1991. Performance analysis of the basic timestamp ordering algorithm via Markov modeling. *Perform. Eval. 12,* 17–41.

SINGHAL, V. AND SMITH, A. J. 1997. Analysis of locking behavior in three real database systems. *Int. J. Very Large Data Bases 6,* 40–52.

SKARRA, A. H. AND ZDONIK, S. B. 1989. Concurrency control and object-oriented databases. In *Object-Oriented Concepts, Databases, and Applications.* W. Kim and F. H. Lochovski, Eds., ACM Press, New York, 395–421.

SRINIVASAN, V. AND CAREY, M. J. 1993. Performance of B+ tree concurrency control algorithms. *Int. J. Very Large Data Bases 2,* 361–406.

TAY, Y. C. 1987. *Locking Performance in Centralized Databases.* Academic Press, Orlando, FL.

TAY, Y. C. 1990. Issues in modeling locking performance. In *Stochastic Analysis of Computer and Communication Systems,* H. Takagi, Ed., North-Holland, New York, 631–658.

TAY, Y. C., GOODMAN, N., AND SURI, R. 1985a. Locking performance in centralized databases. *ACM Trans. Database Syst. 10,* 4 (Dec.), 415–462.

TAY, Y. C., SURI, R., AND GOODMAN, N. 1985b. A mean value performance model for locking in databases: The no-waiting case. *J. ACM 32,* 3 (July), 618–651.

THOMASIAN, A. 1982. An iterative solution to the queueing network model of a DBMS with dynamic locking. In *Proceedings of the Thirteenth Computer Measurement Group Conference* (San Diego, Dec.), 252–261.

THOMASIAN, A. 1985. Performance evaluation of centralized databases with static locking. *IEEE Trans. Softw. Eng. 11,* 2 (April), 346–355.

THOMASIAN, A. 1988. Distributed optimistic concurrency control methods for high-performance transaction processing. *IEEE Trans. Knowl. Eng. 10,* 1 (Jan./Feb.), 2–18.

THOMASIAN, A. 1992. Performance analysis of locking policies with limited wait depth. In *Proceedings of Performance 92 and ACM SIGMETRICS Joint Conference* (Newport, RI, June), 115–127.

THOMASIAN, A. 1993. Two-phase locking and its thrashing behavior. *ACM Trans. Database Syst. 18,* 4 (Dec.), 579–625.

THOMASIAN, A. 1994. A fractional data allocation method for distributed databases. In *Proceedings of the Third International Conference on Parallel and Distributed Information Systems* (Austin, TX, Sept.), 168–175.

THOMASIAN, A. 1995a. Performance analysis of locking policies with limited wait depth. *Perform. Eval.* (submitted). Also IBM Res. Rep. RC 19977, Hawthorne, NY, March.

THOMASIAN, A. 1995b. Checkpointing for optimistic concurrency control methods. *IEEE Trans. Knowl. Data Eng. 7,* 2 (April), 332–339.

THOMASIAN, A. 1996a. *Database Concurrency Control: Methods, Performance, Analyses.* Kluwer Academic, Boston.

THOMASIAN, A. 1996b. On realistic modeling and analysis of lock contention. *Inf. Syst. 21,* 5, 409–430.

THOMASIAN, A. 1997. A performance comparison of locking policies with limited wait depth. *IEEE Trans. Knowl. Data Eng. 8,* 2 (May/June), 421–434.

THOMASIAN, A. 1998. Distributed optimistic concurrency control methods for high performance transaction processing. *IEEE Trans. Knowl. Data Eng. 10,* 1 (Jan/Feb.), 2–18.

THOMASIAN, A. AND NICOLA, V. 1993. Performance evaluation of a threshold policy for scheduling readers and writers. *IEEE Trans. Computers 42,* 1 (Jan.), 83–98.

THOMASIAN, A. AND RAHM, E. 1990. A new distributed optimistic concurrency control method and a comparison of its performance with two-phase locking. In *Proceedings of 1990 International Conference on Distributed Computing Systems* (Paris, May), 294–301.

THOMASIAN, A. AND RYU, I. K. 1983. A decomposition solution to the queueing network model of the centralized DBMS with static locking. In *Proceedings of the ACM SIGMETRICS Conference* (Minneapolis, MN, Aug.), 82–92.

THOMASIAN, A. AND RYU, I. K. 1986. Performance comparison of concurrency control methods for shared centralized databases. Unpublished rep., IBM T. J. Watson Research Center, Hawthorne, NY.

THOMASIAN, A. AND RYU, I. K. 1989. A recursive solution method to analyze the performance of static locking systems. *IEEE Trans. Softw. Eng. 15,* 10 (Oct.), 1147–1156.

THOMASIAN, A. AND RYU, I. K. 1991. Performance analysis of two-phase locking. *IEEE Trans. Softw. Eng. 17,* 5 (May), 386–402.

WEIHL, W. E. 1988. Commutativity based CC for abstract data types. *IEEE Trans. Computers 37,* 12 (Dec.), 1488–1505.

WEIKUM, G. 1991. Principles and realization strategies of multilevel transaction management. *ACM Trans. Database Syst. 16,* 4 (March), 132–180.

WEIKUM, G., HASSE, C., MOENKEBERG, A., AND ZABBACK, P. 1994. The COMFORT automatic tuning project. *Inf. Syst. 19,* 5, 381–432.

YU, P. S., DIAS, D. M., AND LAVENBERG, S. S. 1993. On modeling database concurrency control. *J. ACM 40,* 4 (Sept.), 831–872.

Received March 1995; revised March 1997; final revision accepted November 1997