Centralized Concurrency Control Methods for High-End TP

Alexander Thomasian
IBM T. J. Watson Research Center
P. O. Box 704
Yorktown Heights, NY 10598, USA

1. Introduction

The purpose of this survey is to review recent developments in the area of concurrency control (CC) in centralized transaction processing (TP) systems and their performance evaluation. The topics covered in this article are as follows. Firstly, we discuss the performance implications of some basic CC methods. Secondly, we describe some recently introduced CC methods, which are suited to very high throughput TP systems with potentially high data contention. Thirdly, we discuss some recent performance studies of CC methods, with emphasis on analytic solutions. Only simple (flat) transactions with a single commit point, which are commonly used in high-volume TP are considered here [Date83, BeHG87, GrRe91]. Thus we do not consider transaction parallelism [CaLi88], "long-lived" transactions for computer-aided design [KoKB90], sagas [GaSa87], and other transaction models [GrRe91].

A large number of CC methods have been proposed which are usually classified into three categories:

1. locking methods,
2. optimistic methods based on certification or validation,
3. timestamp ordering methods [Date83, Bern87]

Two-phase locking (2PL) and its variants (see Section 2) remain the prevalent CC method [Date83, Bern87, GrRe91]. Performance studies of locking based CC methods in centralized systems (up to 1986) are reviewed in [TayY87] (see Section 4), the emphasis of our discussion is therefore on more recent studies. Optimistic methods proposed in [KuRo81] are of great theoretical interest, but may be unsuitable from an implementation viewpoint [Haer84].

The timestamp ordering method, which has been proposed for distributed systems, assures that the serialization order is the same as transaction timestamps assigned to them at the beginning of their execution. For example, an "older" transaction is aborted when it attempts to read a data item written by a "younger" transaction. Aborted transactions are assigned a new timestamp when they are restarted. Timestamp ordering is obviously applicable to centralized systems and an analysis of its performance appears in [Sing91]. This method, however, has a performance inferior to others in a centralized system. For example, it is shown in [RyTh90] that it is outperformed by the immediate restart locking method [AgCL87]. Timestamp ordering is therefore not considered in further discussions.

Data contention is an inevitable by-product of the CC method associated with the DBMS of a TP system. The performance of a TP system is in fact often dominated by other factors, which are briefly discussed here The following discussion is in the context of high-end TP on a centralized computer system, i.e., a mainframe with shared everything (main memory and disks) architecture.

A key factor impacting performance is the application and system software pathlength. Especially the latter may vary significantly from system to system depending on functionality, e.g., referential integrity checking in relational databases [Date83]. Transaction pathlength determines the cost of running transactions and conversely the maximum throughput attainable by a system. For an intended transaction throughput the rate of disk accesses is usually matched by a sufficient number of disk arms to keep disk utilization at an acceptably low level (say below 30% for random accesses). Lock or data contention adversely affects the performance of a TP system in the form of CC overhead (e.g., lock acquisition, deadlock detection, etc.), and transaction blocking and restarts.

The requirement for increased transaction throughput (T) results in an increase in the mean number of active transactions in the system ($\bar{M}$), since $\bar{M} = TR$, where R denotes the mean transaction response time. The probability of lock conflict (p_c) increases linearly with $\bar{M}$ according to the lock contention model postu-

lated in [GHOK81] (see Section 2), which is adopted in most performance studies of locking. It is shown in Section 2 that the level of lock contention in a TP system with 2PL is proportional to p_c. As T and hence $\bar{M}$ is increased, the performance of a TP system might not scale with increased hardware resources, unless precautionary measures are taken to alleviate the potential lock contention bottleneck. This problem is aggravated by the trend towards more complex transactions, which lead to longer response times. It is not uncommon to break a long transaction to a sequence of very short transactions for performance purposes. The semantics of these "short" transactions may be incompatible with the application requirements, in which case user level intervention is required, e.g., manually cancel one reservation if a second related reservation cannot be made, which would be handled routinely otherwise according to the transaction atomicity paradigm [Bern87, GeRe91].

Transaction response time is mainly affected by the number of disk accesses. This number can be reduced significantly by providing a database buffer in main memory. The buffer hit ratio is also affected by the replacement policy. As the cost of random access memories is decreasing, it is economically justifiable to maintain more and more data in main memory. In the extreme, main storage databases are adopted to achieve high volume TP for short transactions (as in IMS Fastpath). A discussion of various issues in main storage databases appears in [Eich89], including CC which is less of a problem in this case. We envision here a high performance centralized TP system with a large database buffer in main memory, but disk accesses are still necessary when buffer misses occur.

Transaction response time is also affected by its commit time and more generally the recovery algorithm. We assume a recovery paradigm suited for high-performance TP systems [Bern87, GrRe91], such as *steal/no-force*, which means dirty pages in the buffer might be replaced (and hence written onto disk) and that pages modified by a transaction need not be written onto disk as part of transaction commit. Since the log disk may become a bottleneck, the number of disk writes can be reduced by using a group commit paradigm, i.e., committing multiple transactions with one disk write [GrRe91]. The provision of non-volatile memories (as achieved by battery backup in the case of power failures) makes it possible to commit a transaction as soon as log data is written into the non-volatile memory. The contents of this memory may be written to the log disk using relatively large block sizes for the sake of efficiency. In the case of restart-oriented CC methods, special provisions such as maintaining a private workspace or maintaining the undo log [Bern87] of currently active transactions in main memory is required for reducing abort overhead.

When the degree of lock contention is low, the CC method does not have much affect on performance, since there is no transaction blocking and there are no restarts to resolve data conflicts. This statement is predicated on the assumption that the various CC methods entail the same overhead, which might not be generally true. Separating policy from correctness in concurrency control design suggests however that different CC methods can be handled in the same framework and present comparable overheads [Robi84]. Optimistic CC methods have the additional overhead of maintaining a private workspace and externalizing updates when the transaction is committed [Haer84].

There is the additional trade-off between the granularity of locking and the degree of lock contention, e.g., a coarser granularity of locking may be appropriate to reduce the overhead of lock acquisition at the cost of increased lock contention. We concern ourselves with relatively fine locking granularity required for high volume transaction processing, e.g., record rather than page level locking which has major implications on recovery [MHLPS89]. We do not concern ourselves with the lock contention between short update transactions and long (mainly) read-only queries. *Strict 2PL* (all exclusive locks are held until transaction completion time) is postulated here [Bern87]. A level 3 (repeatable read) consistency level [Date83], which requires that shared locks be held until commit time tends to be unsuitable for mainly read-only queries being processed in a TP environment. A level 2 consistency (cursor stability) [Date83] results in a significant reduction in lock contention, since a shared lock is maintained only while an object is being accessed. Cursor stability allows non-2PL and also non-serializable schedules, which is not acceptable for all applications.

The lock manager of the DBMS may use specialized locking methods to reduce the affect of lock contention in processing indexes and other data structures [Bern87, GrRe91]. The escrow locking paradigm [O'Ne86] may also be used in conjunction with hot-spots in the form of aggregate data. The CC methods described in Sections 3 and 4 of this paper are then in addition and orthogonal to lock contention reduction methods already in place in the DBMS.

In addition to shared everything systems, parallel and distributed systems with shared nothing and shared disk architectures are also quite important for TP as well as query processing. The additional source of complexity in shared disk systems is database buffer

or cache coherence and coordination of data page and CC information transfers between the nodes of the system [MoNa91]. 2PL can be extended to a shared nothing environment by adding a two-phase commit protocol [Bern87], but distributed deadlock detection remains a problem, such that timeouts are usually used to resolve deadlocks. The timeout interval is a difficult parameter to determine [JeTK90]. Conversely, some CC methods proposed for distributed systems such as timestamp-ordering and the wound-wait and wait-die methods [Bern87] are also applicable to centralized systems (after some simplifications).

The review is organized as follows. In Section 2, we discuss the performance of 2PL with emphasis on its thrashing behavior. We also present two recent studies dealing with load control in 2PL to prevent thrashing. In Section 3, we discuss a property called *access invariance,* which serves as the basis of optimistic and several newly proposed CC methods. In Section 4, a set of locking methods which are based on limiting the wait-depth of blocked transactions in locking, including the Wait-Depth Limited (WDL) CC method are presented. Conclusions appear in Section 5.

2. Limitations of Two-Phase Locking

The performance of 2PL has been an area of intense investigation (see Chapter 4 in [TayY87]). In this section after describing the general behavior of a TP system with 2PL, we proceed to discuss a recent analysis of 2PL performance and its thrashing behavior, which has been observed in numerous earlier studies [Thom82, FrRo85, TaGS85, TayY87, AgCL87]. We also briefly review analytic solution methods for 2PL.

In *static locking* all locks are pre-claimed and a transaction is not activated until it has acquired all of the locks it requires for its execution. Transactions in the system may be activated in strict first-come-first-served (FCFS) order, but significant improvements in performance are possible with a *FCFS with skip* policy, which allows deviations from FCFS by allowing a transaction to be activated as soon as it can acquire all of its required locks [ThRy83]. Locks are assigned to transactions by an *atomic action* to prevent deadlocks. Static locking tends to be unrealistic since the objects required by a transaction are not known a priori (without executing the transaction), unless the transaction is being rerun (see Section 3) or in batch systems where a coarse granularity of locking (at a file rather than a page or record level) is being used.

In *dynamic locking* the locks required by a transaction are requested on demand, such that deadlocks involving two or more transactions are possible. The performance of a transaction processing system with dynamic locking (referred hereafter to as 2PL for the sake of brevity) can be affected by varying the number of transactions *activated* (M) at the computer system. Activated transactions commence their execution, while there may be other transactions at the system awaiting activation. Restricting the number of activated transactions serves the role of load control, the need for which is justified below. In 2PL, a transaction encountering a lock conflict is blocked, unless there is a deadlock in which case one of the transactions involved in the deadlock cycle is restarted. Thus, only a subset of activated transactions are active and the rest of them are blocked. As M is increased in a system with adequate main memory, the system throughput initially increases, but there is a degree of transaction concurrency ($\tilde{M}$) at which the bottleneck resource in the system (say the CPU) becomes saturated.

As the processing capacity and the main storage of the system is increased to accommodate higher transaction throughputs, $\tilde{M}$ at which the CPU saturates also increases, but depending on the degree of lock contention in the system it is possible for the throughput to reach a peak at $\hat{M} \leq \tilde{M}$, beyond which the throughput does not increase and may even drop sharply. This effect which is referred to as *thrashing* is due to the increase in transaction blocking, i.e., as M is increased there is a point beyond which the number of transactions active in the system starts *decreasing.* In the limit when the TP system is thrashing, there are only a few active transactions and the computer system is highly under-utilized. Thrashing can be attributed to the fact that blocked transactions in the system hold locks and result in further blocking (a snowball effect). In fact the wasted processing due to transaction aborts to resolve deadlocks can be ignored unless the system is highly CPU-bound, since the additional processing tends to be a small fraction of the total processing [Thom91]. This can be ascribed to the fact that deadlocks tend to be rare [GHOK81, ThRy91].

The question arises whether there is a simple way to determine the thrashing point in 2PL. In what follows, we will be more specific and consider M transactions of size k (requesting k locks) such that the lock requests are uniformly distributed over the active lifetime of a transaction, i.e., the transaction consists of $k+1$ identically distributed processing steps (this rather abstract model which is also used in [TaGS85, Thom91], is adequate for our purposes). There are N data items (objects) in the database, and there is a lock associated with each object.

It is observed in [TaGS85, TayY87] that a TP system with infinite resources and 2PL achieves its peak throughput at: $\hat{M}k^2/N \simeq 1.4$. Note that the probabil-

ity of lock conflict when all lock requests are in exclusive mode is: $p_c \simeq (M-1)k/(2N)$ [GHOK81], which is the ratio of locks held (in exclusive mode) by the *other* $M-1$ transactions and the number of locks in the database (each transaction holds $k/2$ on the average when the processing times for transaction steps are equal). Given that the fraction of shared lock requests or accesses to hot spots are given, appropriate expressions can be written for p_c to take these effects into account (see e.g., [TaGS85]). It follows that the mean number of conflicts per transaction is:

$$n_c = 1-(1-p_c)^k \simeq kp_c = (M-1)k^2/2N$$

Thus the degree of lock contention in a system with fixed size transactions as characterized by n_c, which should be smaller than 0.7. Given that the database and transaction characteristics remain fixed, one load control scheme to prevent thrashing is to constrain the number of transactions activated in the system by $\hat{M}$.

This result does not apply however to a TP system with variable transaction sizes, i.e., transactions requesting a variable number of locks. The analysis in [Thom91] expresses the mean waiting time due to lock conflict as:

$$W = W_1(1+0.5\beta+1.5\beta^2+...)$$

where W_1 is the mean waiting time when the lock conflict is with an active transaction and β denotes the fraction of blocked transactions in the system or equivalently, the fraction of time each transaction is blocked, i.e., $\beta = n_cW/R$ where R denotes the mean transaction response time. The above equation is based on the following two approximations: the probability of blocking (resp. mean waiting time) at level $i > 1$ in the wait-for tree (level zero corresponds to active transactions) is $P_b(i) = \beta^{i-1}$ (resp. $W_i = (i-0.5)W_1$). Defining $\alpha = n_cA$ with $A = W_1/R$ and summing the series in the parenthesis yields a cubic equation in β. This equation has a feasible solution ($\beta < 1$) as long as $\alpha \leq 0.226$ and the system is unstable (thrashing) otherwise. Also $\beta = 0.378$ when $\alpha = 0.226$.

In fact the parameter α reflects the effect that the variability of transactions size has on performance degradation in 2PL [Thom91, RyTh91]. Firstly, $n_c = K_1p_c$ where p_c is proportional to the mean number of locks (L) held by transactions, which is a function of the second moment of the number of requested locks and hence its variance. Also W_1 is a function of the third moment of transaction size. For geometrically versus fixed size transactions (with the same mean size) there is a factor of 2 increase in L and a factor of 3 in $A = W_1/R$, which results in a factor of 6 increase in α. It follows that treating variable size transactions as fixed size transactions with the same mean transaction size tends to yield optimistic results. The opposite is also possible, e.g., unintentionally fixing the probability that a transaction requests another lock in a simulation concerned with fixed size transactions leads to higher than intended lock contention levels due to a geometric distribution for the number of locks!

The variability of transaction size has a major effect on the probability of deadlocks and the analysis in [GHOK81] is extended in [ThRy91] to study this effect. This analysis also introduces a correction to the previous analysis by noting that for a deadlock to be possible the transaction holding the requested lock must be in the blocked state at the time the other transaction requests the lock (based on the previous discussion the fraction of time a transaction is in the blocked state is β). Simulation results have shown the analysis to be quite accurate in predicting the probability of deadlock as long as the system is not thrashing ($\alpha < 0.226$) by just considering deadlocks of cycle length two. The analysis shows that the probability of deadlock for geometrically distributed transactions is an order of magnitude higher than for fixed size transactions.

A TP system with 2PL is susceptible to thrashing at higher lock contention levels (as characterized by α). This probability becomes non-negligible, however, only when α approaches its maximum value. Even at $\alpha = 0.226$, a TP system can operate for a long time at this load before thrashing occurs [Thom91]. In an open system, the randomness of the arrival process causes fluctuations in the number of transactions in the system which may result in increases in the value of α and this in turn may lead to thrashing. Thrashing can be prevented by restricting the number of transactions that are *activated* in the system. Even a system which is load controlled in this manner is susceptible to thrashing when it is operating at a high value for α. One factor affecting thrashing in a closed system is the variability in transaction size. In a system with distinct transaction classes (as determined for example by the database area accessed by them) the number of transactions that can be activated in each class can be restricted by associating classes with a fixed number of regions in which they can be activated. Such load control is available in commercial TP monitors (see e.g., [MoWe91].

Two recent studies have discussed methods to cope with thrashing in 2PL. The first of these studies proposes the "half-and-half" rule which states that the system is susceptible to thrashing when only 50% of the transactions activated in the system are active and they are mature (hold over 25% of the locks they need) [CaKL90]. This is consistent with the conclusions of the analytic study in [Thom91] where it was determined

that the fraction of blocked transactions is a good indicator of the onset of thrashing. A more recent study is based on the conflict rate, which is defined as the ratio of the number of locks held by all transactions to the number held by active transactions [MoWe91]. (Special adjustments need to be made for counting shared locks). Extensive experimentation indicates that the system approaches thrashing at a conflict-rate of 1.3. Simulation results in [MoWe91] seem to demonstrate that it reacts more quickly to overload conditions than the "half-and-half" method.

Analytic methods for analyzing TP systems with 2PL can be classified into the following categories:

1. Analytic solutions based on *queueing network models* which utilize pseudo-servers to represent the delay introduced by lock contention (see e.g., [Thom82]). There is a pseudo-server associated with each object (lock) in the database, but due to symmetry they can be aggregated into a single "flow-equivalent server." The routing probability to this server is determined by the probability of lock conflict and otherwise the transaction proceeds with its execution. The "service time" at the pseudo-servers is determined by the pattern according to which locks are requested during the execution time of the transaction, since this affects their holding time. For example, if locks are requested uniformly during the processing time of the transaction this delay is one third of transaction response time. The variability in transaction size (number of requested locks) also affect this parameter as was indicated earlier. Since the service time at the pseudo-servers is a function of transaction response time, which is not known a priori, an iterative solution is required.

2. The analytic solution method in [TaGS85] applies to fixed size transactions. A "flow diagram" is used to represent the progress of an individual transaction as it proceeds through its execution steps. There are two sets of states depending on whether the transaction is active or blocked. The durations of the active states of a transaction are assumed to be known a priori. The mean holding time in blocked states is the mean waiting time for a lock to be released (W). Estimating W is the most difficult aspect of the analysis and requires numerous assumptions to make the analysis tractable. The analysis in [TaGS85] yields the same probability for all states, which is due to the fact that deadlocks are ignored [HsZh87] (the probability of deadlock increases rapidly with the number of locks that it holds [RyTh90a, ThRy91]).

   The mean number of active transactions is expressed as the mean of its bounds ($M(1-n_c/3) < M_a < M(1-n_c/6)$) in [TaGS85, TayY87] as: $M_a = M(1-n_c/F)$ with $F = 4.5$, which is corrected to $F = 4$ in further work by the authors. This expression is tantamount to $W = R/2$ which is an over (resp. under) -estimate at low (resp. high) lock contention levels.

   The upper bound is applicable at lower lock contention levels, when almost all lock conflicts are with active transactions. Referring back to the analysis in [Thom91] in effect we have $W \simeq W_1$ and $\beta \simeq \alpha$ such that $M_a = M(1-\beta) \simeq M(1-\alpha)$. In the case of fixed size transactions $\alpha = A(M-1)k^2/2N$ with $A \simeq 1/3$, while in the case of variable size transactions the required parameters should be estimated for the given distribution of transaction sizes. Also in the case of an open system with Poisson arrivals (say with rate λ), given that the mean transaction residence time in the system without lock contention is $r(\lambda)$ then its mean response time is given by $R(\lambda) = r(\lambda)/(1-\alpha)$. Since p_c is a function of the mean number of transactions in the system: $\bar{M} = \lambda R(\lambda)$, the above equation yields a quadratic equation in $R(\lambda)$, whose negative root is the acceptable solution [ThRy91].

3. The mean value analysis in [Thom91] which applies to variable size transactions with identically distributed step durations which was outlined above.

   The analysis in [ThRy91] allows variable transaction sizes and different processing times for transaction steps, which may be obtainable from an underlying queueing network model. Only two levels of transaction blocking: with currently active transactions and transactions which are blocked by an active transaction are considered. Simulation results show that this analysis is quite accurate up to relatively high lock contention levels, but cannot predict peak throughput.

4. Analytic solution based on Markov chain representations of the state of a system, rather than an individual transaction (as in [TaGS85]). An appropriate state representation should be adopted in this case to avoid an explosion in the size of the state space, but this is at the cost of complications in analyzing the transition rates among the states. The state representation used in [RyTh90A] is simply the number of active transactions (J). The holding time in each state is determined by an underlying computer system model. Transitions occur when a transaction requests a lock or is completed. A lock request may result in the transition

$J \rightarrow J$, $J \rightarrow J-1$, and $J \rightarrow I, I > J$ depending on whether the lock request is successful, unsuccessful with blocking, and unsuccessful with an abort (the locks released by the activated transaction result in the activation of others). This analytic method is based on a multilevel solution, which is quite straightforward and modular. It is also based on very few approximations as compared to [TaGS85, Thom91]. Note that this analysis takes into account deadlocks, while the other analyses do not.

As noted in [TaGS85] separation of hardware and data contention is a very nice property which leads to simplifications in the analysis. This technique is not applicable, however, when there is interaction between hardware resource and lock contention. For example, extra processing is required in static locking to check for the eligibility of transactions blocked in a queue for activation [ThRy83] or to take into account the extra processing required to block and later reactivate a transaction encountering a lock conflict.

3. Access Invariance and its Use in High Contention Environments

Serialization is the correctness criterion in TP, i.e., the concurrent execution of a set of transactions has the same effect as some serial execution and that the ordering of transaction executions is not stipulated [Bern87]. It has been conjectured that with a very high probability a transaction will perform the same set of operations and access the same set of objects when it is executed before or after a small set of other transactions (of the order of level of concurrency in the system) [FrRT89, FrRT90]. The property that a transaction accesses the same set of objects is referred to as *logical access invariance* [FrRT89, FrRT90]. Since accesses to the database are in the form of blocks or pages, there is also *physical access invariance,* i.e., transactions access objects co-residing with the primary object. In effect when the transaction is re-executed the buffer has been primed.

Access invariance is the basis of the *two-phase transaction processing method,* proposed in [FrRT89, FrRT90], according to which a transaction is executed in two phases. During the first phase also referred to as *virtual execution,* no CC method is exercised and the execution of the transaction results in the fetching of blocks from disk, which are not already available in the database buffer. Standard locking (2PL) or some other CC method can then be used in the second phase, which tends to be very short since no disk IO is required (provided that access invariance prevails). Since the transaction residence time in the system tends to be an order of magnitude smaller in the second execution phase than in the first, there is also an order of magnitude reduction in the effective level of transaction concurrency and hence the probability of lock conflict (p_c). In the absence of full access invariance disk accesses may be required and this incurs a performance penalty, but correctness (serializability) is not compromised since a CC method is in effect in this phase.

The two-phase processing method has the disadvantage that it basically requires that a transaction be executed (at least) twice. The processing required for the second phase can be much smaller than the first one in a TP system specialized for this purpose. In any case, the second phase requires less CPU processing, since no disk accesses are required. In fact a CC method can be used in the first phase and provided that a transaction does not encounter a data conflict (explained below) it can be completed at the end of the first phase without requiring the second phase. This requires the design of a phase-dependent policy to handle four types of conflicts [FrRT90]:

1. conflicts among first phase transactions
2. conflicts among second phase transactions
3. a first phase transaction issues a request that conflicts with a second phase transaction
4. a second phase transaction issues a request that conflicts with a first phase transaction

Obviously various combinations of CC methods can be used for the two phases, several of which are considered in [FrRT89, FrRT90]. First let us consider 2PL in both phases, but allow locks in the second phase to have a higher preemptive priority with respect to locks held by first phase transactions. The other three types of lock conflicts in the above list are handled through blocking unless there is a deadlock. Upon the preemption of a lock held by a first phase transaction there are two choices:

1. abort it, but this will entail further conflicts unless the restart of the transaction is appropriately delayed;
2. the transaction releases all of its locks but continues its processing in virtual execution mode.

The optimistic CC method also relies heavily on access invariance. Transaction are run without requesting locks and updates by transactions are reflected in a private workspace. Upon the completion of a transaction, the CC manager checks if the database objects read by the transaction are up-to-date (this can be easily accomplished by associating timestamps or version numbers with each object and modifying it when the objects is updated). A transaction succeeding its validation commits and externalizes updated values from its private workspace, otherwise the transaction is restarted.

The forementioned method is referred to as the optimistic die policy [FrRT89, FrRT90] or silent commit [RyTh87].

The wasteful processing incurred by transactions failing their validation can be reduced by notifying transactions that the objects read by them have been updated by committed transactions. This can be attained by transactions posting pseudo-locks or access entries when they read an object [FrRT89, FrRT90], which can be used to identify conflicted transactions. This policy is referred to as the optimistic kill policy [FrRT89, FrRT90] or broadcast commit [RyTh87].

In accordance with the two-phase processing paradigm, it is advisable to adopt the die rather than kill policy in a high lock contention environment when disk accesses contribute heavily to transaction response time. In the case of a main storage resident database the kill policy is the obvious choice, since there is no advantage to be gained by prefetching the data. In the presence of access invariance the overall policy should be die/kill, i.e., kill in all phases after the first one in case the transaction is restarted more than twice. That this is a good policy was verified by simulation studies reported in [FrRT89, FrRT90].

The optimistic die policy in the first phase can also be combined with locking in the second phase. Conflicts between first and second phase transaction are resolved by giving priority to lock requests. This is advantageous because transactions in the second phase are almost assured to complete their execution. Static locking or lock preclaiming is also possible in this case because the identities of objects to be referenced are known as a result of the first execution phase [FrRT89, FrRT90]. In fact, the choice of the CC method for second phase tends to have little impact on performance, but locking methods are preferable since they prevent a transaction from being restarted more than once [FrRT90]. This lock preclaiming scheme is applicable to a shared-nothing system where deadlocks can be prevented by preclaiming locks at all nodes in the same order [ThRa90].

Previous analytic studies of optimistic CC are reviewed in [RyTh87]. This paper also presents the analysis of optimistic die and kill methods. The key point in the analysis is that transactions encounter conflicts at random instants of time, i.e., according to a Poisson process. This analytic study, which allows multiple transaction classes (e.g., based on transaction size), has served as the starting point for numerous other studies.

A problem with optimistic CC in particular, and two-phase processing methods based on access invariance, in general, is that the wasted processing increases quadratically with transaction size. This is hence referred to as the *quadratic effect* [FrRT89, FrRT91]. For example, in the case of the optimistic die protocol the probability that a transaction fails its validation is proportional to the number of objects it accesses (say k) and the time it spends in the system, which is also proportional to k. The effect that a large fraction of wasted processing is due to longer transactions was observed in [RyTh87] and the simulation studies in [FeRT89, FrRT90]. Thus it is not surprising that optimistic CC methods outperform 2PL in systems with infinite resources [FrRo85, AgCL87]), but that 2PL outperforms optimistic CC methods in a system with finite resources [AgCL87]. It is thus important to develop CC methods which outperform 2PL, but limit the additional processing caused by transaction restarts.

4. Locking Methods Allowing Limited Wait-Depth

Most implementations of CC methods in TP systems are based on 2PL, which is constrained due to transaction blocking. It is therefore important to consider modifications of 2PL in seeking a method with improved performance. Several studies have concerned themselves with limiting the wait-depth of blocked transactions as a means of improving the performance of locking methods.

An extreme policy is to abort a transaction as soon as it encounters a lock conflict, which is referred to as the no-waiting [TaSG85] or the immediate restart policy [AgCL87]. There are two possibilities to replace the aborted transaction with another one immediately (referred to as *fake restart* or *resampling of locks* or rather lock requests) or maintaining the identities of transaction locks (*no resampling*) but delaying its restart to avoid cyclic restarts or livelocks of two or more transactions. Transaction delay may be selected from an appropriate random distribution (say uniform) with a mean equaling the mean transaction response time (see e.g., [TaSG85, AgCL87]) or until the conflicting transactions have left the system [RyTh90B] (referred to as *restart waiting* in [FrRT91]). In comparing the performance of 2PL and no-waiting in [TaSG85, TayY87] an infinite resource model is postulated with fake restart, which leads to the conclusion that the no-waiting method is superior to 2PL (except in a narrow region where 2PL outperforms the no-waiting policy by at most 5%). In fact this is not so and in a system with finite resources, 2PL tends to outperform the no-waiting method [AgCL87, RyTh89]. Also 2PL tends to outperform the no-waiting method with no resampling of locks regardless of hardware resource contention [RyTh89].

The cautious waiting policy [HsZh87] aborts a transaction encountering a lock conflict with another blocked transaction. Thus T_2 in the wait chain $T_2 \rightarrow T_1 \rightarrow T_0$ is aborted when it requests a lock held by T_1. Note that restarting T_2 while T_1 is still blocked will lead to repeated restarts and wasted processing. It is observed in [HsZh87] that this method outperforms the no-waiting policy, because it is beneficial to wait rather than restart a transaction in terms of reducing transaction response time (even if the system has infinite resources and wasted processing can be tolerated).

The running priority method aborts transaction T_1 in the wait-for graph $T_2 \rightarrow T_1 \rightarrow T_0$, when T_2 requests a lock held by T_1 thus giving priority to an active transaction [FrRo85]. In *symmetric* running priority T_1 is aborted regardless of the order in which the lock conflicts occurred [FrRT89, FrRT91]. Furthermore the restart of T_1 may be delayed pending the completion of T_0 and T_2, according to the restart waiting paradigm. It can be argued that running priority outperforms cautious waiting by increasing the number of active transactions in the system.

The Wait-Depth Limited (WDL) method in addition to limiting the wait-depth, judiciously selects the transaction to be aborted to minimize (reduce) wasted processing. There is a *length function* associated with each transaction which reflects the processing acquired by the transaction, the number of locks held by the transaction, etc. One of the few rules in WDL(1) (which limits the wait depth to one) is to restart the transaction T_1 in $T_2 \rightarrow T_1 \rightarrow T_0$ *unless* T_1 is longer than T_0 and all other transactions like T_2 which are blocked by it. Simulation results have shown that WDL outperforms 2PL, running priority and even optimistic methods in high MIPS but finite resource systems (in an infinite resource system optimistic CC, which conforms to the essential blocking paradigm [FrRO85] outperforms WDL).

In addition while WDL aborts transactions much more frequently than there are aborts to resolve deadlocks in 2PL, simulation results in [FrR89] have shown that it demonstrates almost the same behavior as 2PL in a high data contention system with slow processors (up to the point that 2PL achieves its peak throughput). Also WDL allows much higher throughputs than possible with 2PL, at the cost of little extra software complexity to decide which transaction if any needs to be restarted and additional processing due to transaction aborts and restarts.

As far as the analysis of such locking methods is concerned, the method adopted in [TaSG85] for the no-waiting method is most appropriate. It is based on considering the progress made by each transaction in isolation, while taking into account the effect that other transactions have on the target transaction. This is in effect a semi-Markov chain such that the duration of each transaction step is allowed to be general, although a Markov chain representation with exponential holding times is required to analyze other CC methods. This method is successfully applied to the analysis of cautious waiting in [HsZh87] and the timestamp ordering method in [Sing91].

5. Conclusions

The field of CC method in centralized systems is relatively mature. A large number of CC methods have been proposed and their performance has been compared through analytic and simulation studies. Unfortunately most analytic solutions of CC methods do not yield explicit expressions for global performance measures, therefore only numerical comparisons of the performance of CC methods are possible.

Two-phase locking with fine (record-level) granularity, combined with specialized locking methods for hot spots and indexes seems to be adequate to cope with lock contention problems (see e.g., [GrRe91]). Other methods based on optimistic CC may be unacceptable due to implementation considerations and the additional CPU processing they incur. Methods such as wait-depth limited (WDL) [FRRT89, FRRT91] are in fact orthogonal to the forementioned schemes to reduce lock contention (e.g., record versus page level locking) and may also be considered as alternatives. This is because WDL requires relatively little modifications to existing software, incurs a relatively small run time overhead, although it requires additional system processing capacity to attain higher throughputs than possible with 2PL.

A very large number of distributed CC methods have been proposed, but there has been little work to compare their performance [CaLi88]. The locking methods discussed in this paper can be extended to work in a distributed environment, e.g., the optimistic/die method followed by lock preclaiming [ThRa90] and the distributed wait-depth limited method [FHRT91].

Some recommendations for future work (in a centralized TP system) are as follows.

1. The "flat" transaction model has been used in most performance studies. An investigation of other models is required.
2. The database access model is also quite simple and is usually based on exclusive/shared access to abstract database objects. Much more complex lock request patterns are generated by relational queries, which need to be incorporated in perfor-

mance studies, e.g., consider an update statement which selects a subset of the records in a table before updating them.

3. More work remains to be done to better understand the mechanisms leading to lock conflicts. Also it is important to determine the effect of varying the workload on lock contention. For example, in most performance studies, it is assumed that the effective database size remains fixed as the transaction arrival rate is varied, which may not be necessarily true.

4. Further work remains to be done in the area of load control and transaction scheduling to reduce lock contention and to prevent thrashing.

5. Performance issues related to CC in active databases, real-time transactions, object-oriented databases require further investigation. New CC methods may be warranted in these environments.

Acknowledgements

Close collaboration with Peter Franaszek and John Robinson has greatly affected the contents of this article. I also wish to thank Hank Korth and Dick Muntz for their comments to improve the quality of presentation.

References

[AbGM88] R. Abbott and H. Garcia-Molina. "Scheduling real-time transactions: A performance evaluation" *Proc. 14th Int'l Conf. on Very Large Data Bases,* Los Angeles, CA, August 1988, pp. 1-12.

[KoKB90] H. F. Korth, W. Kim, and F. Bancilhon. "On long duration CAD transactions," in *Readings in Object-Oriented Databases,* S. Zdonik and D. Maier (eds.), Morgan-Kaufman, San Mateo, CA, 1990.

[AgCL87] R. Agrawal, M. J. Carey, and M. Livny. "Concurrency control performance modeling: Alternatives and implications," *ACM Trans. Database Systems 12,*4 December 1987, 609-654.

[CaKL90] M. J. Carey, S. Krishnamurthy, and M. Livny. "Load control for locking: the 'half and half' approach," *Proc. 9th ACM Symp. on the Principles of Database Systems (PODS),* Nashville, TN, April 1990, pp. 72-84.

[CaLi88] M. J. Carey and M. Livny. "Distributed concurrency control performance: A study of algorithms, distribution and replication," *Proc. 14th Int'l Conf. on Very Large Data Bases,* Los Angeles, CA, August 1988, pp. 13-25.

[Date83] C. J. Date. *An Introduction to Database Systems: Vol. II,* Addison-Wesley, Reading, MA, 1983.

[Daya88] U. Dayal. "Active database management systems," *Proc. 3rd Int'l Conf. on Data and Knowledge Bases,* Jerusalem, Isreal, June 1988, pp. 150-169.

[Eich89] M. H. Eich. "Main memory database research directions," in *Database Machines: Sixth Int'l Workshop IWDM '89 Proc.* H. Boral and P Faudemay (eds.), Springer Verlag, 1989, pp. 251-268.

[FrRo85] P. A. Franaszek and J. T. Robinson. "Limitations of concurrency in transaction processing." *ACM Trans. Database Systems 10,*1 (March 1985), 1-28.

[FrRT89] P. A. Franaszek, J. T. Robinson, and A. Thomasian. "Concurrency control for high contention environments," *IBM Research Report RC 14665,* Hawthorne, NY, June 1989 (to appear, *ACM Trans. Database Systems).*

[FrRT90] P. A. Franaszek, J. T. Robinson, and A. Thomasian. "Access invariance and its use in high contention environments," *Proc. 6th Int'l Conf. on Data Engineering,* Los Angeles, CA, February 1990, pp. 47-55.

[FrRT91] P. A. Franaszek, J. T. Robinson, and A. Thomasian. "Wait depth limited concurrency control," *Proc. 7th Int'l Conf. on Data Engineering,* Kobe, Japan, April 1991, pp. 92-101.

[FHRT91] P. A. Franaszek, J. R. Haritsa, J. T. Robinson, and A. Thomasian. "Distributed concurrency control based on limited wait-depth," *IBM Research Report RC 16881,* Hawthorne, NY, May 1991.

[GMSa87] H. Garcia-Molina and K. Salem. "Sagas," *Proc. 1987 SIGMOD Annual Conf.* San Francisco, CA, May 1987, pp. 249-259.

[GHOK81] J. N. Gray, P. Homan, R. L. Obermarck, and H. Korth. "A straw-man analysis of waiting and deadlock," *IBM Research Center Report RJ 3066,* San Jose, CA, February 1981 (abstract in *Proc. 5th Berkeley Workshop on Distributed Data Management and Computer Networks,* Berkeley, CA, February 1981, p. 125).

[GrRe91] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*, Morgan-Kaufman, San Mateo, CA, 1991.

[Haer84] T. Haerder. "Observations on optimistic concurrency control schemes," *Information Systems 9,2* (October 1984), 287-317.

[HsZh87] M. Hsu and B. Zhang. "The mean value approach to performance evaluation of cautious waiting," Technical Report, Aiken Computation Laboratory, Harvard University, February 1987.

[JeTK90] B. C. Jenq, B. C. Twitchel, and T. W. Keller. "Locking performance in a shared nothing parallel database machine," *IEEE Trans. on Knowledge and Data Eng. 1,4* (December 1989), 178-198.

[KuRo81] H. T. Kung and J. T. Robinson. "On optimistic concurrency control methods," *ACM Trans. on Database Systems 6,2* (June 1981), 213-226.

[MoWe91] A. Moenkeberg and G. Weikum. "Conflict-driven load control for the avoidance of data-contention thrashing," *Proc. 7th Int'l Conf. on Data Engineering*, Kobe, Japan, April 1991, pp. 632-639.

[MHLPS89] C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh, and P. Schwarz. "ARIES: A transaction recovery method supporting fine locking and partial rollbacks using write-ahead logging," *IBM Research report RJ 6649*, Almaden, CA, January 1989 (to appear, *ACM Trans. Database Systems)*.

[MoNa91] C. Mohan and I. Narang. "Recovery and coherency control protocols for fast inter-system page transfer and fine granularity locking in a shared disks transaction environment," *Proc. 17th VLDB Conf.*, Barcelona, Spain, Sept. 1991.

[O'Ne86] P. E. O'Neil. "The escrow transaction method," *ACM Trans. Database Systems 11,4* (December 1986), 405-430.

[Robi82] J. T. Robinson. "Separating policy from correctness in concurrency control design," *Software Practice and Experience 14,9* (September 1984), 827-844.

[RyTh87] I. K. Ryu and A. Thomasian. "Performance analysis of centralized databases with optimistic concurrency control," *Performance Evaluation 7,3* (1987), 195-211.

[RyTh90A] I. K. Ryu and A. Thomasian. "Analysis of database performance with dynamic locking," *Journal of the ACM 37,3* (July 1990), 491-523.

[RyTh90B] I. K. Ryu and A. Thomasian. "Performance analysis of dynamic locking with the no-waiting policy," *IEEE Trans. Software Engineering 16,7* (July 1990), 684-698.

[Sing91] M. Singhal. "Performance analysis of the basic timestamp ordering algorithm via Markov modeling," *Performance Evaluation 12* (1991), 17-41.

[TaSG85] Y. C. Tay, R Suri, and N. Goodman. "A mean value performance model for locking in databases: The no-waiting case," *Journal of the ACM 32,3* (July 1985), 618-651.

[TaGS85] Y. C. Tay, N. Goodman, and R. Suri. "Locking performance in centralized databases," *ACM Trans. Database Systems 10,4* (December 1985), 415-462.

[TayY87] Y. C. Tay. *Locking Performance in Centralized Databases*, Academic Press, Orlando, FL, 1987.

[Thom82] A. Thomasian. "An iterative solution to the queueing network model of a DBMS with dynamic locking," *Proc. 13th Computer Measurement Group Conf.*, San Diego, CA, December 1982, pp. 252-261.

[ThRy83] A. Thomasian and I. K. Ryu. "A decomposition solution to the queueing network model of a centralized DBMS with static locking," *Proc. ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems.*, Minneapolis, MN, August 1983, pp. 82-92.

[ThRa90] A. Thomasian and E. Rahm. "A new distributed optimistic concurrency control method and a comparison of its performance with two-phase locking," *Proc. 10th Intl. Distributed Computing Conf.*, Paris, France, May 1990, pp. 294-301.

[Thom91] A. Thomasian. "Performance limits of two-phase locking," *Proc. 7th Int'l Conf. on Data Engineering*, Kobe, Japan, April 1991, pp. 426-435.

[ThRy91] A. Thomasian and I. K. Ryu. "Performance analysis of two-phase locking," *IEEE Trans. Software Engineering SE-17,5* (May 1991), 386-402.