Modeling and Analysis of Transaction Processing Systems

Philip S. Yu

IBM Research Division, T. J. Watson Research Center
Yorktown Heights, NY 10598

Abstract. In recent years, the demand for on-line transaction processing systems has grown rapidly with ever stringent performance requirements. In this paper, we examine several issues encountered in designing transaction processing systems and report some of the recent advancements in analytical performance modelling methodology on analyzing alternative design trade-offs. First of all, the Concurrency Control (CC) scheme employed can profoundly affect the performance of transaction processing systems. A general analytic modelling approach is presented that can be applied to analyze the various CC schemes under a unified framework, including locking, various optimistic schemes, and hybrid schemes. The analysis can capture the effect of skewed data access, different lock modes, variable length transactions and the buffer retention effect on rerun transactions. Next we consider the analysis of buffer hit probability. In a multi-node environment, whether in a cluster or client server environment, buffer coherency needs to be addressed. The cross invalidation phenomenon can have an adverse effect on the buffer hit probability. A general methodology to analyze various coherency control schemes is examined to predict the buffer hit probability. A hierarchical approach is used to decompose the modelling of transaction processing systems into three components: hardware resource, concurrency control and buffer models. The interaction among the components is then captured through a fixed point iteration.

1. Introduction

In recent years, the demand for on-line transaction processing systems has grown rapidly with ever stringent performance requirement. With the advent of VLSI technology, coupling multiple micro-processors to support high transaction rates has been pursued by various vendors [KRON86, YU92B,C]. Furthermore, client-server architectures have increasingly become a common approach to support transaction processing [KIM90, DEUX90, HORN87]. There has also been considerable interest in geographically distributed transaction processing systems, in which the databases may be distributed among regional systems [GRAY86]. In this paper, we examine several issues encountered in designing transaction processing systems to accommodate these requirements and report some of the recent advancement in analytical modelling methodology on analyzing performance trade-offs of alternative designs to address these issues.

First of all, the Concurrency Control (CC) scheme employed can profoundly affect the performance of transaction processing systems. This is particularly so as the demand for transaction throughput increases leading to greater data contention. There have been numerous analytical studies of the performance of CC schemes. A survey of the analysis of locking in centralized databases can be found in [TAY84,TAY90], while [SEVC83] examines early analytical work on the performance of different CC schemes in distributed databases. More recent work includes [CHES83, MORR85, TAY85A,B, THOM85, YU85, RYU87, YU87, CELL88, SING88, HSU88,

DAN88, HART89, CICI90A,B, SING91, THOM91, YU91, CICI92, HSU92, YU92A,93]. While there have been numerous simulation studies of concurrency control performance, we do not mention them here since the focus of this paper is on analytical methodology. Here a general analytic modelling framework is presented which can be applied to analyze the performance trade-offs of various CC schemes under a unified methodology, including locking, various optimistic schemes, and hybrid schemes. This is based on the methodology developed in [YU93] which is derived and extended from a number of specific studies we have done, e.g. [YU85, CORN86, YU87, CICI92,90A,B, YU90,91,92A]. The analysis can capture the effect of skewed data access, different lock modes, and variable length transactions.

Next we consider the analysis of buffer hit probability. In a multi-node environment, whether in a cluster or client-server environment, buffer coherency control needs to be addressed. The cross invalidation phenomenon can have an adverse effect on the buffer hit probability. In the literature, there have been few studies based on analytic models for this. Recently, a general methodology has been developed in [DAN90,91,92A,93A,B] to model the effects of various coherency control schemes on buffer hit probability. Here we present the buffer modeling methodology based on these works. We note that coherency control can be provided by generalizing the CC manager function to track the buffer contents [DIAS89]. Various lock retention schemes have also been proposed to support coherency control [RAHM86, RAHM88, MOHA91, DAN92A,93A, WILK90, FRAN92, CARE91], although most of the performance studies are based on simulations.

The execution time of a transaction depends on three main factors: 1) the concurrency control protocol used for resolving conflict in accessing data pages (waiting, abort etc.), 2) the buffer hit probability that determines the number of I/O operations to be performed by the transaction, and 3) the processing time and the queueing delay in accessing system resources such as CPU, etc.. We model the concurrency control, buffer hit probability and system resource access times separately and capture their interactions via a higher level model. This higher level model relates quantities from the lower level models through a set of non-linear equations. The solution of the higher level model corresponds to the solution of a fixed point problem which we solve through an iterative process. The transaction execution time depends on the buffer hit probability estimated by the buffer model, and by the queueing and services estimated by the CPU model. The CC model estimates the data conflict probability based on the transaction execution time, and this in turn affects both the buffer and resource models.

In Section 2 we discuss the workload model. In Section 3 we discuss the general methodology for analyzing concurrency control schemes. We apply this methodology in Section 4 to analyze various CC schemes. For optimistic schemes the effect of changed buffer hit probability when a transaction is rerun is handled in the analysis. For locking we provide a simple approximate expression for the mean lock waiting time. In Section 5 we present the methodology to analyze the buffer hit probability under various coherency control schemes. In Section 6, generalization of the methodology in Section 5 is considered. Concluding remarks appear in Section 7.

2. Workload Model

In this section, we briefly discuss the issue on how to provide characterization of database workload. There are two entities involved. One is the transaction and the other is the database. The transactions can generally be characterized by the number of granules accessed, the mode of each access (i.e. share vs exclusive), the instructions executed between granule accesses, etc. The database is characterized by its size, i.e. the number of granules in the database. If the system studied is in operation, all these parameters can often be measured from some tracing facilities.

To model the progression of a transaction, our transaction model can be described as follows. Assume that N_L is the number of granules accessed by a transaction. The transaction model would consist of $N_L + 2$ states or stages, (We use the term granule to refer to the unit of data to which concurrency control is applied.) The transaction has an initial setup phase (including program fetch, and message processing), state 0. Following the initial setup, a transaction progresses to states 1, 2, ... , N_L, in that order. This is the execution phase. At the start of each state, i, the transaction begins to access a new granule and moves to state $i + 1$ when the next new granule access begins. At the end of state N_L, if successful, the transaction enters into the commit phase at state $N_L + 1$.

We still need to specify the granule accessed at each state. This is where the complexity of the workload characterization lies. We make the following assumption which is used in obtaining all the analytical results:

- All granule access requests are independent. While it is reasonable to assume that granule access requests from different transactions are independent, independence cannot hold within a transaction if a transaction's granule accesses are distinct. However, we use this assumption as an approximation for the granule accesses from the same transaction. If the probability of accessing any particular granule is small, e.g. when the number of granules in the database is large and the the number of granules accessed by each transaction is small, this approximation should be very accurate. A similar approximation is also made in [TAY84,85A,B].

(Note that this assumption is certainly not appropriate for query scanning through a large portion of the databases sequentially.) The frequencies that granules accessed by transactions are generally skewed as observed in real workload analysis [DAN93C]. An example of a skewed access pattern often used in the literature is the 80-20 access rule where 80% of the references access 20% of the granules (i.e., the hot set), while the remaining 20% of the references access the rest of of the granules (i.e., the cold set).

Here we use a generalized skewed access model which covers the 80-20 access pattern as a special case. In this model, based on the frequency of data access, the data granules are logically grouped into M partitions,[1] such that the probability of accessing any granule within a partition is uniform. Let β_i denote the fractional size of partition i. Let α_i denote the probability that any data access lies in partition i. An example of this type of logical grouping is the 80-20

[1] This logical grouping is done for computation purpose only, since granules with the same frequency of access has the same buffer hit probability under the LRU replacement policy.

access pattern where $K = 2$, $(\alpha_1,\alpha_2) = (0.8, 0.2)$, and $(\beta_1,\beta_2) = (0.2, 0.8)$. The uniform access model is another special case with $M = 1$, $\alpha_1 = 1$ and $\beta_1 = 1$. Even if each transaction application follows the 80-20 access pattern on some relations, multiple transaction applications may have different hot sets and cold sets, and different access rates. Overall this can result in a larger number of logical partitions with different access frequencies and partition sizes. In [DAN93C], real workload traces are analyzed to fit this kind of skewed access model. It was found that with a small number of partitions, the matching of the buffer hit prediction based on this model with the trace driven simulation can be very close.

We note that skewed access pattern has different effects on data contention and buffer hit probability [DAN90B]. The presence of skew increases the data contention probability. This would have a negative effect on the the transaction response time. On the other hand, the skewed accesses improve the buffer hit probability. This would have a positive effect on the response time. However, in a multi-node environment, the skewed accesses also makes the buffer invalidation effect more severe, and thus negatively affects the buffer hit probability. The analytic modelling methodology would need to capture all these effects.

3. Concurrency Control: General Methodology

In an environment with no data contention, the transaction response time is determined by the queueing and processing delay in accessing hardware resources such as CPU, I/O, etc.. In the presence of data contention, the transaction response time further depends upon the occurrence of data conflict in accessing the database. The probability of data conflict for any transaction depends not only on the CC scheme itself but also on the transaction response time which in turn depends on the conflict probability. For example, when locking is used, if the lock contention probability increases, the transaction response time increases due to additional lock waits. In turn, longer transaction response time leads to a longer lock holding time, and hence to a higher lock contention probability. Similarly, when an optimistic CC scheme is used if the transaction abort rate increases, there is a concomitant increase in the CPU utilization causing longer response and data holding times, and therefore a higher probability of abort. We model hardware resource access times and the effect of CC separately and then we solve the models simultaneously using an iteration to estimate the mean transaction response time. We assume here an open model with Poisson transaction arrivals, but the analysis can be extended to a closed model as shown in [YU93]. The hardware resource contention can be modeled by conventional queueing models with CPU servers and disk servers. Since this is rather straightforward, we will only briefly outline it for the case of analyzing optimistic CC (Section 4.1.2).

In this and the next section, we will concentrate on how to estimate the data conflict probability and mean lock waiting time. A model for predicting buffer hit probability can also be incorporated to provide a more accurate estimate of the number of IO's and will be discussed in Section 5.

3.1 Conflict Analysis

A **conflict** is defined to be an event in which a transaction accesses a data granule that is currently accessed or in use by another transaction in an incompatible mode. The result of a conflict is either a transaction wait or transaction abort. We assume that there is only one transaction type. (Generalization to multiple transaction types is straightforward and can be found in

[YU93].) We also make the following additional assumptions and then show how they can be relaxed at the end of this section:

- The granule access distribution is uniform over the set of granules.
- All accesses to granules are exclusive or update.

Let λ be the transaction arrival rate, N_L be the mean number of granule accesses by each transaction, and L be the number of granules in the database. Let T_H denote the mean holding time of a granule, that is the period of time from when the granule is first accessed until the end of the transaction.

Consider a generic locking scheme. We assume that the probability of deadlock is negligible compared to lock contention as shown in [GRAY81, TAY85B, YU93]. Before a granule can be accessed, a transaction needs to acquire a lock on that granule. A lock request can be made right before each granule is accessed as in dynamic locking, or all lock requests can be made at the beginning of a transaction as in static locking. Each time a lock request is made the corresponding entry in the lock table is examined. If no other transactions hold an incompatible lock on that entry, the granule is locked and the access is granted. The granule is locked until it is released by the transaction. If the granule has already been locked in an incompatible mode, lock contention occurs. (Note that for now we are considering only exclusive granule access so that all accesses to the same granule are incompatible.) Due to the assumption that deadlocks do not occur and the assumption of uniform granule access distribution, the arrival rate of lock requests for a particular granule is $\lambda N_L/L$. We have

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

$$P_W = \frac{\lambda N_L T_H}{L} \quad , \tag{3.1}$$

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

$$P_{CONT} \leq 1 - (1 - \frac{\lambda N_L T_H}{L})^{N_L} \quad . \tag{3.2}$$

The derivation is straightforward for the case of fixed length transactions. As shown in [YU93], this is also true for the case of variable length transactions.

We will use the upper bound in Equation (3.2) as an approximation to P_{CONT}, i.e.

$$P_{CONT} \approx 1 - (1 - \frac{\lambda N_L T_H}{L})^{N_L} \quad . \tag{3.3}$$

An $O(1/L)$ approximation to Equation (3.3) is given by

$$P_{CONT} \approx \frac{\lambda N_L{}^2 T_H}{L} \quad . \tag{3.4}$$

We next consider a generic Optimistic CC (OCC) scheme. Transactions access granules as they progress. At the end of execution, if all granules accessed are the up-to-date versions, the transaction will commit and reflect the updated values into the database.[2] In this generic OCC scheme, the commit is assumed to be instantaneous. The effect of non-zero commit time is addressed in the next section when specific certification schemes are considered. At the end of the commit phase, for each granule updated, any transaction accessing the granule is notified that the granule is invalid. Transactions accessing invalid granules are aborted at commit time (referred to as pure OCC). (Later we will consider the case where transactions accessing invalid granules are aborted immediately, referred to as broadcast OCC.) The rate of invalidation to a particular granule is equal to the transaction throughput multiplied by the mean number of granules updated per transaction and divided by the database size, i.e. it is given by $\lambda N_L/L$. From [YU93], we have

Conflict Probability under OCC: Assume that the invalidation times for a given granule form a Poisson process and that all such processes (to different granules) are independent. Then the probability, P_A, that a transaction is aborted when it first tries to commit is upper bounded as follows:

$$P_A \leq 1 - exp\left(-\frac{\lambda N_L^2 T_H}{L}\right) \tag{3.5}$$

where in Equation (3.5) T_H is the mean granule holding time for transactions during their first run only.

Note that Equation (3.5) need not hold for rerun transactions since the mean length of rerun transactions need not equal that of first run transactions except if all transactions have the same fixed length. We will use the upper bound in Equation (3.5) as an approximation to P_A, i.e.

$$P_A \approx 1 - exp\left(-\frac{\lambda N_L^2 T_H}{L}\right) . \tag{3.6}$$

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

$$P_A \approx \frac{\lambda N_L^2 T_H}{L} . \tag{3.7}$$

We next consider the case of non-uniform accesses to the database. Assume that the database consists of multiple (M) sets or partitions of granules with α_i fraction of the references go to β_i fraction of the granules. Let T_{H_i} be the mean granule holding time for partition i. The lock utilization for partition i is $\lambda\alpha_i N_L T_{H_i}/\beta_i L$. Assume Poisson arrivals of lock requests and that lock contention events for a transaction are independent, a similar expression can be obtained as in (3.3). In the same way that the approximation in Equation (3.4) was shown to follow from the expression, it follows that

[2] We assume that during the course of a transaction any updates are kept in a private buffer, and are only made visible to other transactions after a successful commit, as in [BERN87, KUNG81].

$$P_{CONT} \approx \sum_{i=1}^{M} \frac{\lambda \alpha_i^2 N_L^2 T_{H_i}}{\beta_i L} \quad . \tag{3.8}$$

As we assume that each lock request independently references partition i with probability α_i, the mean lock holding times are the same for each partition i, i.e. $T_{H_i} = T_H$, $1 \le i \le M$. Under OCC in this environment, a similar development to that for a single partition of granules gives,

$$P_A \approx \sum_{i=1}^{M} \frac{\lambda \alpha_i^2 N_L^2 T_{H_i}}{\beta_i L} \quad . \tag{3.9}$$

Finally we illustrate how the case of different access modes can be handled by considering the case of exclusive and shared modes. Assume that each data access has probability p^u of being in exclusive mode and that the mode of each access is independent. Again we consider the case of uniform accesses to the database. First consider the generic locking case, similar to Equation (3.3), we can get,

$$P_{CONT} \le 1 - (1 - \frac{\lambda N_L T_H}{L})^{N_L p^u} (1 - \frac{\lambda N_L p^u T_H}{L})^{N_L(1-p^u)} \quad .$$

An approximation similar to that in Equation (3.4) is,

$$P_{CONT} \approx \frac{\lambda N_L^2 T_H p^u (2 - p^u)}{L} \quad . \tag{3.10}$$

Under the OCC environment, a similar development leads to

$$P_A \approx \frac{\lambda N_L^2 T_H p^u}{L} \quad . \tag{3.11}$$

3.2 *Conservation Property of Conflict*

The approximate expressions for the transaction abort/invalidation probability when using the generic OCC scheme given in Equation (3.7) and for the transaction contention probability when using the generic locking CC scheme given in Equation (3.4) are the same for the case of a single transaction type with exclusive access mode. (These approximations contain only the dominant term, which is $O(1/L)$ in the approximations in Equations (3.6) and (3.3) respectively. However, we believe that the $O(1/L)$ expressions provide insight into the effects of data conflicts for the various CC schemes we consider, and thus we discuss them in this section.) This is true even if the database consists of multiple partitions of granules with different access probabilities as shown in Equations (3.8) and (3.9). This is referred to as the **conservation property of conflict** in [YU93]. Note, however, that the mean granule holding times, T_H, will in general not be equal for the two schemes, as discussed later in this section, so that the conflict probabilities will differ.

In the optimistic case, the abort probability affects the number of reruns and hence the CPU utilization which affects T_H. In the dynamic locking case, the contention probability increases the transaction response time, and consequently, the mean granule holding time, T_H. If the CPU resource is unlimited, the optimistic approach provides lower conflict since T_H is close to the processing time. If the CPU resource is very limited, static locking tends to provide lower

conflict. (In static locking, a transaction tries to acquire all locks at once at the beginning. If it cannot get all of them, it does not hold on to any locks, but tries later or waits until they all become available.) In this case, the wait time to get the locks does not enlarge the granule holding time. In [YU90], a CC scheme using a combination of locking and OCC is proposed, and is shown to improve the performance over both OCC and locking schemes by striking a balance between the effect of transaction aborts and lock waits as discussed in Section 4.3.1.

Finally, a word of caution. If we allow for different access modes like shared and exclusive, the conservation property may no longer hold. This is clear by comparing Equations (3.10) and (3.11). This is due to the fact that the concept of access modes is a further optimization to reduce conflicts between transactions and different CC schemes like locking and OCC, depending upon the specific implementation, can have different capabilities to exploit it. Another case where the conservation property is violated is when transactions which may not be successfully committed abort conflicting transactions, or when transactions wait for subsequently aborted transactions, as in wound wait [ROSE78], locking with no-waiting [TAY85A], various running priority schemes in [FRA85], and wait depth limited concurrency control in [FRA92]. These schemes produce unnecessary aborts or waits. On the other hand, these schemes may result in smaller values for T_H compared to locking.

4. Analysis of Various CC schemes

In this section, we demonstrate how to apply the methodology just presented to approximate the mean response time for optimistic and locking CC schemes. Unless stated otherwise, for simplicity we will assume that there is only one transaction type, the granule access distribution is uniform and all accesses are exclusive. Relaxing these constraints can be similarly addressed as in Section 3. Furthermore, only the case of fixed length transactions is considered and a centralized lock manager is assumed. Generalization of the methodology to handle variable length transactions and distributed environments with database replications can be found in [YU93].

4.1 Optimistic Protocols

In this subsection, we consider the OCC schemes. Various OCC schemes have been proposed in the literatures. Two commonly considered variations are the **pure OCC** scheme where a transaction is aborted only at certification time, and the **broadcast OCC** scheme where a transaction is aborted as soon as any granule it has accessed is made obsolete by a committing transaction. Most previous analyses on OCC schemes ignored the fact that due to buffering in main memory rerun transactions may not need to re-access from disk all the granules brought in during previous runs and therefore came out in favor of broadcast OCC. In [YU93], the analysis specifically captures this **buffer retention** effect and is able to show that pure OCC can in fact outperform broadcast OCC. A combination of the pure OCC and broadcast OCC, referred to as **OCC with broadcast during rerun** is proposed and analyzed in [YU92A]. In this scheme, a transaction uses pure OCC for its first run, and employs the early abort of broadcast OCC for any subsequent reruns. Compared with pure OCC, the immediate abort during rerun

reduces wasted CPU instruction, hence improves the performance. In the following, we present the analysis for the OCC with broadcast during rerun to illustrate the methodology. The analysis of the pure OCC follows directly from that of the first run of this scheme. The analysis of the broadcast OCC can be generalized from that of the rerun transactions, albeit using a more complex set of difference equations [YU93].

4.1.1 OCC with Broadcast During Rerun

In Section 3 when we analyzed the pure OCC scheme we assumed that commit processing was instantaneous. We now approximate the abort probability in a more realistic setting. We assume that each transaction at the beginning of each state of its execution phase informs the CC manager of its request to access to a new granule. The CC manager keeps a list of transactions accessing each granule, and also maintains locks for granules held by transactions in commit. During the first run, if a transaction requests access to a granule for which an incompatible lock is held by a transaction in commit, it is marked for abort. At commit time, the CC manager checks if a transaction has been marked for abort. If so, the CC manager removes the transaction from the list of granules accessed, and the transaction restarts after a backoff interval with mean duration $T_{Backoff}$. Otherwise, the transaction enters commit processing, and the CC manager grants locks for the granules accessed by the transaction and marks for abort any transactions that have conflicting access. The transaction then writes commit records to the log and propagates the updates to the disk, modeled as taking a mean time of T_{Commit}, following which locks are released. Let N_L denote the (fixed) number of granules accessed by a transaction. The initial setup phase of the transaction consists of execution of a mean of P_{INPL} instructions and a mean of I_{INPL} I/O's. In the first run of a transaction, the mean time in each state i, $1 \le i \le N_L$, is assumed to be the same and is denoted by $\hat{R}$. $\hat{R}$ corresponds to execution of a mean of $\hat{P}$ instructions, and a mean of $(1 - H)$ I/Os, where H denotes the (buffer hit) probability that the accessed granule is found in a main memory buffer.

Assuming instantaneous commit processing, Equation (3.6) approximates the invalidation probability on the first run P_A as $1 - exp(-\lambda N_L^2 T_H / L)$, where T_H is the mean granule holding time and is equal to $(N_L + 1)\hat{R}/2$. This corresponds to the probability that a transaction is marked for abort by a transaction that enters commit processing after the invalidated transaction accessed a conflicting data granule. Recall that transactions are also aborted when the new granule accessed at any stage is already locked by a transaction in commit; the probability that this occurs is approximated by Equation (3.3) with T_{Commit} replacing T_H. Transactions (in the process of commit) that hold a lock on the new granule requested and transactions that initiate commit during the i-th state of the transaction being considered are different sets of transactions. We assume that invalidations due to these two cases are independent events since they arise from conflicts from different sources. Thus, the probability of transaction abort is approximated as,

$$P_A \approx 1 - exp(-\lambda N_L^2 T_H / L)\ (1 - \frac{\lambda N_L T_{Commit}}{L})^{N_L} . \quad (4.1)$$

This can be further approximated as,

$$P_A \approx \frac{\lambda N_L^2 (T_H + T_{Commit})}{L} . \quad (4.2)$$

While we can derive Equation (4.2) from Equation (4.1), what is interesting to note is that we can directly write down the expression by adding together the conflict terms from the two different sources, i.e. conflicting with transactions already in commit, and aborting by transactions initiating commit. This is referred to as the **additive approximation** property in [YU93]. In doing this, it is important to identify the sources of abort. In this case there are two: one is from invalidation and the other is from lock contention. Taking advantage of the additive approximation, we can estimate each term separately based on Equations (3.4) and (3.7) and sum them.

We next consider the rerun transactions. After state $N_L + 1$, if the transaction is aborted, it returns to state 1. During reruns, upon entering state i, the transaction not only informs the concurrency control manager of its access to the i-th granules, but also checks to see if it has already been marked for abort. If already marked for abort, the transaction is immediately aborted and returns to state 1.

To analyze the abort probability of rerun transactions, recall that all read I/Os are assumed to be done in the first run; hence, there is no I/O during rerun. The mean time at each state i, $1 \le i \le N_L$, is assumed to be the same and is denoted by R' corresponding to execution of P' instructions. The probability of abort $A(i)$ in the i-th state of a rerun is estimated as,

$$A(i) = 1 - exp(-iN_L\lambda R'/L)\ (1 - \frac{N_L \lambda T_{Commit}}{L}). \quad (4.3)$$

The remaining response time at the start of a rerun (if any) is estimated as,

$$B = \sum_{i=1}^{N_L} \{\prod_{k=1}^{i-1}(1 - A(k))\} A(i) \{B + T_{Backoff} + (i-1)R'\} + \{\prod_{k=1}^{N_L}(1 - A(k))\} N_L R' . \quad (4.4)$$

In the first summation term of Equation (4.4), $\{\prod_{k=1}^{i-1}(1 - A(k))\}A(i)$ corresponds to the abort probability after advancing to state i in a rerun and $\{B + T_{Backoff} + (i-1)R'\}$ is the expected response time measured at the start of this rerun. In the second term of (4.4), $\prod_{k=1}^{N_L}(1 - A(k))$ is the

probability that a rerun completes and commits, while $N_L R'$ is the corresponding response time. The expected remaining instructions executed by a rerun transaction is estimated as,

$$D = \sum_{i=1}^{N_L} \{\prod_{k=1}^{i-1}(1 - A(k))\}A(i)\{D + T_{Backoff} + (i-1)P'\} + \{\prod_{k=1}^{N_L}(1 - A(k))\}N_L P' . \quad (4.5)$$

This equation is similar to (4.4) with B replaced by D, and R' by P'.

4.1.2 Hardware Resource Model

We now describe the hardware resource model and show how it can be coupled with the data contention model through an iteration. We assume that the system consists of a single CPU and a database that is spread over multiple disks. Assuming Poisson arrivals of transactions, we model the processors as an M/M/1 queue with FCFS discipline and the disks as an infinite server. Other open queueing network models could be used; we chose this one for simplicity. The processor utilization is given by

$$\rho = \frac{\lambda \times \{P_A D + P_{INPL} + N_L \hat{P}\}}{MIPS} \quad (4.6)$$

where MIPS is the processor speed. Then the mean times in different states are given by as,

$$\begin{aligned} R_{INPL} &= \frac{P_{INPL}}{MIPS(1-\rho)} + I_{INPL} \times IOTIME \\ \hat{R} &= \frac{\hat{P}}{MIPS(1-\rho)} + \hat{I} \times IOTIME \\ R' &= \frac{P'}{MIPS(1-\rho)} \end{aligned} \quad (4.7)$$

where $\hat{I} = 1 - H$ is the average number of I/Os per state in the first run (respectively any subsequent run). We will discuss how to estimate H in Section 6. The overall mean transaction response time is given by

$$R = R_{INPL} + N_L \hat{R} + T_{Commit} + P_A(T_{Backoff} + B). \quad (4.8)$$

From Section 4.1.1, we know P_A and $A(i)$ depend upon $\hat{R}$ and R', respectively, whereas $\hat{R}$ and R' depend upon the CPU utilization ρ which in turn depends upon P_A and $A(i)$. There clearly is an interdependency. We can first pick some arbitrarily small abort probabilities, P_A and $A(i)$, and calculate the mean response times, $\hat{R}$ and R'. The mean response times are then used to calculate a new set of abort probabilities. The process continues and typically converges in a few iterations.

4.2 Standard Locking

We now show how the methodology can be applied to analyze the standard two-phase locking (2PL) scheme. Let N_L be the (fixed) number of granules accessed by a transaction. In our transaction model, transactions make lock requests at the beginning of states 1, ..., N_L. If the granule has already been locked in an incompatible mode, the transaction is enqueued at the CC manager and waits till the lock becomes available. As before for locking, we neglect the probability of deadlock. For locking each state i, $1 \le i \le N_L$, is divided into two substates $\bar{i}$ and $\hat{i}$. In substate $\bar{i}$ the transaction holds $i-1$ locks and is waiting for its i-th lock request to be satisfied. In substate $\hat{i}$ it holds i locks and is executing. Let a denote the mean time in substate $\hat{i}$ which is assumed to be independent of i and is obtained from the hardware resource model as discussed later in this section. The probability that substate $\bar{i}$ is entered upon leaving substate $\hat{i}-1$ is the lock contention probability for a lock request, P_W, which is given in Equation (3.1) and assumed to be independent of i. The time spent in substate $\bar{i}$, given it is entered, is the lock waiting time whose mean we denote by R_W which is also assumed to be independent of i. The unconditional mean time in substate $\bar{i}$ is therefore $b = P_W R_W$. R_W can be approximated as follows. The mean number of transactions in substate $\bar{i}$ (respectively $\hat{i}$) is λb (respectively λa). Since a transaction in substate $\bar{i}$ (respectively $\hat{i}$) holds $i-1$ locks and is waiting for its i-th lock (respectively holds i locks) and the granule access distribution is uniform, the probability that a lock request contends with a transaction in substate $\bar{i}$ (respectively $\hat{i}$) is $(i-1)\lambda b/L$ (respectively $i\lambda a/L$), $1 \le i \le N_L$. Similarly, the probability that a lock request contends with a transaction in state N_L+1 is $\lambda N_L c/L$ where $c = T_{Commit}$. Let

$$G = \{\sum_{i=1}^{N_L}(ia + (i-1)b)\} + N_L c \quad . \tag{4.9}$$

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

$$R_W \approx \sum_{i=1}^{N_L}\left(\frac{(i-1)b}{G}\{\frac{R_W}{f_1} + a + s_i\} + \frac{ia}{G}\{\frac{a}{f_2} + s_i\}\right) + \frac{N_L c}{G}\{\frac{c}{f_3}\}. \tag{4.10}$$

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

$$s_i = (N_L - i)(a+b) + c \quad . \tag{4.11}$$

The quantity R_W/f_1 (respectively a/f_2) is the mean remaining time in substate $\bar{i}$ (respectively $\hat{i}$) given that the transaction contended with was in that state, $1 \le i \le N_L$, and similarly for c/f_3. Note that in obtaining Equation (4.10) we have assumed that when a transaction T_1 encounters contention for a lock held by another transaction T_2, then no other transaction T_3 can be waiting for this same lock, i.e. the queue length for any lock never exceeds two. Thus, when T_2 commits and releases the lock T_1 acquires it without further waiting. Note, however, that T_2 can be waiting for a lock held by yet another transaction T_4, and T_4 can also be waiting, etc. Thus, wait chains can build up due to waits for different locks, and this is captured by the analysis. Since transactions typically hold several locks, wait chains of this type (rather than for the same granule) predominate. The factors f_1, f_2 and f_3 depend on the distributions of the times in substates $\bar{i}$, $\hat{i}$ and N_L+1 respectively. Substate $\hat{i}$ corresponds to the CPU execution time and the I/O time if the granule accessed in the state is not found in memory (recall that H denotes the probability that a granule accessed in any state is found in a main memory buffer). Assuming that H is not very close to one and that I/O time is constant and much larger than the processing time (this is true for typical parameter values), then it is easy to show that $f_2 \approx 2(1-H)$. In our studies we will assume that $f_3 = 2$ corresponding to a constant time in the the commit state and that $f_1 = 1$ corresponding to exponential lock waiting times. Comparison with simulations [YU92A] indicated that $f_1 = 1$ yields good results. At low contention levels, the estimates from $f_1 = 1$ are a little pessimistic. Equation (4.10) can be simplified to yield,

$$R_W \approx \frac{\frac{(a+b)^2(N_L+1)(N_L-1)}{6} + \frac{(N_L+1)(\frac{a^2}{f_2}+ab+ac+bc)}{2} + \frac{c^2}{f_3} - ab - bc}{a(\frac{N_L+1}{2}) + c + b(\frac{N_L-1}{2})(1-\frac{1}{f_1})} \quad . \tag{4.12}$$

For large N_L, small c, and small b compared to a, the right side of Equation (4.12) is approximately equal to $((a+b)(N_L-1)+c)/3$. This is the approximation used in [YU87]. Furthermore, it follows from using Equation (3.1) and computing the mean lock holding time T_H from Equation (4.9) that

$$P_W = \frac{\lambda G}{L} \quad . \tag{4.13}$$

The mean response time can be estimated as

$$R = R_{INPL} + R_E + N_L P_W R_W + T_{Commit} \tag{4.14}$$

where $R_E = N_L \times a$ is the mean duration of the execution phase, which can be approximated using a hardware resource model similar to that in Section 4.1.2. Since both R_W and P_W in Equations (4.12) and (4.13), respectively, depend upon $b = P_W R_W$, we need to solve for them through an iteration. Starting with $b = 0$, we can get initial values for R_W and P_W, and thus a new b value. The process continues and typically converges in a few iterations.

4.3 Extensions to Other CC Schemes

4.3.1 Hybrid CC Schemes

Locking tends to suffer form cascade blocking while OCC may suffer from wasting resources due to transaction aborts. Various combinations of locking and OCC schemes have been considered in the literature. One type of hybrid CC schemes is to use pure OCC for the first run and locking for the second run. As the second run will be shorter due to the buffer retention effect, the blocking effect would be minimized. This would also greatly reduce the number of reruns. If static locking is used for the second run, there can be at most one rerun. These types of schemes are analyzed in [YU92A] based on a similar analytic approach presented here.

Another type of hybrid scheme, referred to as locking with deferred blocking, is to divide transaction execution into a non-blocking phase and a blocking phase. At the start of the execution, a transaction is in the non-blocking phase similar to OCC. After a transaction has accessed a predefined number of granules, it tries to enter the blocking phase by obtaining locks on the granules already accessed and if successful, it will request locks for all subsequent granule accesses. This would avoid costly abort at the later states of a transaction execution and avoid long waits as transactions only hold locks at the later states. This scheme is analyzed in [YU90] using similar approach presented here. These hybrid schemes have been shown to lead to better performance than the conventional locking and OCC schemes and are especially attractive for real-time CC [YU94].

4.3.2 Dynamic Timestamp Certification

The OCC schemes considered in Section 4.1 attempt to serialize transactions using certification times as the timestamps [BERN87]. Other OCC certification schemes using dynamically derived timestamps or interval of timestamps have been proposed [BAYE82, BOKS87, YU91]. This type of schemes is effective in reducing the read write conflict. Consider the following scenario. Assume that transaction X reads granule A and transaction Y subsequently updates granule A and commits before transaction X completes. If OCC (pure or broadcast) is used, transaction X will be aborted. This type of conflict is referred to as rw-conflict. However, transaction X can in fact be committed with a back-shifted timestamp earlier than that of transaction Y. This is what the dynamic timestamp certification scheme offers. Note that shifting timestamp backward would not be allowed if it would cause conflict with other committed transactions. For example, if transaction X further updates granule C and some other transaction Z already read the old value of C and committed after transaction Y, giving transaction X a back-shifted timestamp before transaction Y would cause a conflict, referred to as a wr-conflict, with transaction Z and hence violate the serializability constraint.

The methodology presented here has been extended in [YU91] to analyze the dynamic timestamp certification scheme based on timestamp history. In [YU91], a discrete time approach

is adopted to simplify the analysis. A transaction goes through discrete stages (or states) as explained before. We use the mean sojourn time at each state as the basic time unit. The timestamp assigned dynamically at certification time to each transaction can get back-shifted several stages. That is, the back-shift is approximated as a multiple of stages in the analysis. We therefore introduce, pb_i $(i = 0,1, \ldots)$ as the probability that a committed transaction is back-shifted i stages, i.e., the timestamp given to a committing transaction is i stages prior to the actual commit time of the transaction. No back-shift occurs if the transaction has incurred no conflict. (This happens with probability pb_0.) Let P_A be the transaction abort probability,

$$1 = P_A + pb_0 + \sum_{i=1}^{\infty} pb_i \quad . \tag{4.15}$$

The pb_i for an arbitrary transaction X satisfies the following equation

$$pb_i \approx \sum_{j=1}^{\min\{i,L\}} P\{\text{ rw-conflict exactly at stage } L-j+1 \text{ with a committing transaction}\}$$
$$\times P\{\text{conflicting (committing) transaction back-shifted } i-j \text{ stages }\}$$
$$\times P\{\text{ no wr-conflict preventing this transaction backshifting i stages }\} \quad .$$

Assuming transaction Y is the conflicting transaction, the first term in the product represents the probability that a transaction Y committing during stage $L-j+1$ of transaction X has updated a data granule read by transaction X. The second term is the conditional probability that transaction Y needs to be back-shifted $i-j$ stages assuming it can be committed. All rw-conflicts summed up in this equation will lead to the conflicting transaction Y getting a timestamp during the interval of stage $L-i+1$ and thus cause the running transaction X to get a timestamp back-shifted i stages. The first term in the right hand side can be obtained based on the conflict analysis in Section 3. Similarly, the third term can be derived with the back-shift probability taken into account.

For the back-shift distribution of committed transactions, we estimate

$$P\{\text{ conflicting (committing) transaction back-shifted } i-j \text{ stages }\} = \frac{pb_{i-j}}{1-P_A} \quad .$$

Note that the back-shift probabilities have to be normalized by $1-P_A$ because we consider only the case of successful completion in the conditional probabilities.

5. Buffer Coherency Control and Buffer Hit Modelling

We next consider the issue of buffer hit analysis. The problem becomes more complex in a multi-node environment where buffer invalidation can occur. (Here a multi-node environment can mean either a cluster-like environment, or a client-server like environment. The modelling

methodology described below is applicable to both environments, albeit the parameter values in the two environments can be different.) Various buffer coherency control schemes have been proposed in the literature [DIAS89, RAMA89, WILK90, CARE91, DAN93B]. In [DAN93B], these are classified into three approaches: **detection** of invalidated granules, **notification** of invalidated granules, and **propagation** of updated granules. These schemes show different trade-offs on the buffer hit probability and CPU overhead. A general modeling methodology is developed in [DAN93B] to analyze six different coherency schemes under the three approaches.

The buffer hit analysis under the LRU replacement policy can be analyzed based on two simple principles: (1) **conservation of flow on LRU substack composition** and (2) **location content probability proportional to relative push down rate,** as explained later. These have been applied to analyze the single node environment in [DAN90A] and the multi-node environment in [DAN93B, DAN92A,B]. We will use the multi-node environment to illustrate the ideas as the single node case can be considered to be a degenerate case of $N = 1$ in the multi-node environment. Two commonly used coherency control schemes are examined: the check-on-access (CA) and the broadcast or selective invalidation notification (IN) schemes, where CA follows a detection oriented approach and IN uses a notification oriented approach. (Note that the buffer hit probability under the propagation oriented approach is not affected by the the invalidation effect as the invalidated granules are immediately replaced.)

Under the IN approach, an invalidation message containing a list of granules modified by the committing transaction is broadcast to all other nodes during transaction commit [STRI82, DAN90B]. Each node upon receiving the invalidation message will check for the updated granules and mark them invalid if present in its local buffer. This immediately frees up the buffer space occupied by the obsolete granules which are brought to the bottom of the LRU stack. A variation of the scheme is to selectively notified only the nodes with a copy of the invalidated granules

Under the CA approach, the obsolete granules are detected at the granule access time by a transaction. We assume that the coherency control function is integrated with the CC manager/controller [DIAS89] which not only provides the traditional concurrency control service, but also tracks which node has a valid copy of a granule. The lock table entry can contain an additional **valid bit** for each node to indicate whether it has a valid copy. Before accessing any granule, the processing node of the transaction makes a lock request to the integrated CC manager. In response, the integrated CC manager returns not only the requested lock, but also the result of the associated buffer validation check based on the valid bit. Note that if the valid bit is off, it will be turned on after the status of the valid bit is returned as invalidation has now been accomplished. At the lock release time, if the granule has been updated, the valid bits of all other nodes except the updating node (which is the only node with an up-to-date version

of that granule) will be turned off. The CA scheme certainly saves the overhead of sending immediate notification of invalidated granules, but it also reduces the buffer hit probability as the obsolete granules continue to reside in the local buffer.

We consider the case of a homogeneous multi-node environment where the access pattern to the database is the same from all nodes. Generalization to the case where each node has its own affinity data set which is accessed less often by others can be found in [DAN92B, DAN93A,B].

Since the multi-node is assumed to be homogeneous, we focus our attention to a single buffer. We first look at the buffer hit probability under the detection oriented approach (CA), which is more difficult to analyze as both valid and invalid granules are mixed together in the buffer, based on the approach in [DAN93B]. To estimate the steady state probability of buffer hit, we first derive the average number of valid granules of each partition in the local buffer of any node. Let $V_i(j)$ denote the average number of valid granules of partition i in the top j locations of the LRU stack. Therefore, the buffer hit probability of the i-th partition is $h_i = V_i(B)/(\beta_i L)$, and the overall buffer hit probability for a granule requested by a transaction is estimated as

$$H^{ca} = \sum_{i=1}^{M} \alpha_i h_i. \tag{5.1}$$

Let $p_i(j)$ be the probability that the j-th buffer location from the top of the LRU stack contains a granule of partition i. Let $p_i^v(j)$ be the probability that the granule is valid given that the granule belongs to the i-th partition. Then,

$$V_i(j) = \sum_{l=1}^{j} p_i(l) p_i^v(l). \tag{5.2}$$

Also, let $Y_i(j)$ denote the average number of (both valid and invalid) granules of partition i in the top j locations of the LRU stack. Then,

$$Y_i(j) = \sum_{l=1}^{j} p_i(l). \tag{5.3}$$

We will set up a recursive formulation to determine $p_i(j+1)$ and $p_i^v(j+1)$ for $j \geq 1$ given $p_i(l)$ and $p_i^v(l)$ for $l = 1,...,j$. Consider a smaller buffer consisting of the top j locations only. The buffer location $(j+1)$ receives the granule that is pushed down from location j. Let $r_i(j)$ be the rate at which granules of partition i are pushed down from location j. Similarly, let $r_i^v(j)$ be the

rate at which valid granules of partition i are pushed down from location j. Our estimation of $p_i(j)$ and $p_i^v(j)$ are based on the following two observations.

- Conservation of flow: Under steady state conditions, the long term rate at which granules of the i-th partition get pushed down from the top j locations of the buffer equals the rate at which they are brought into the top j locations. Note that if the new access is a hit on an invalid granule present in the top j buffer locations, the granule simply will be refreshed with a valid copy and will not cause a replacement from the top j locations. Hence, the push down rate, $r_i(j)$ is given by

$$r_i(j) = \lambda N_L \alpha_i (1 - \frac{Y_i(j)}{\beta_i L}). \tag{5.4}$$

  Using a similar conservation of flow argument for the valid granules, we equate the long term rate at which valid granules of the i-th partition get pushed down from the top j locations of the buffer, $r_i^v(j)$, to the difference of the rate at which they are brought into the top j locations, and the rate they become invalid. Hence, $r_i^v(j)$ is given by

$$r_i^v(j) = \lambda N_L \alpha_i (1 - \frac{V_i(j)}{\beta_i L}) - (N-1)\lambda N_L \alpha_i p_i^u \frac{V_i(j)}{\beta_i L}. \tag{5.5}$$

- Relative push down rate: The expected value of finding a granule of the i-th partition in the $(j+1)$-st buffer location over all time, $p_i(j+1)$, is approximately the same as the probability of finding a granule of the i-th partition in the $(j+1)$-st buffer location in the event that a granule is pushed down from location j to location $(j+1)$.

$$p_i(j+1) \approx \frac{r_i(j)}{\sum_{l=1}^{M} r_l(j)}. \tag{5.6}$$

  Note that instantaneous value of $r_i(j)$ is dependent on the content of the top j buffer locations, and the more accurate estimation of $p_i(j)$ requires the precise distribution of the content of j buffer locations. Similarly,

$$p_i^v(j+1) \approx \frac{r_i^v(j)}{r_i(j)}, \quad j = 1...B-1. \tag{5.7}$$

Equations (5.2)-(5.7) can be solved iteratively, with the base condition of $p_i(1) = \alpha_i$ and $p_i^v(1) = 1$. Note that, although $r_i(j)$ is a function of the transaction rate (λ), $p_i(j)$, $p_i^v(j)$ and therefore, h_i and H^{ca} are independent of λ, because λ cancels out in Equations (5.6) and (5.7).

Let H^{bi} be the buffer hit probabilities under IN. H^{bi} can be estimated in a similar way as H^{ca} by setting the push down rate, $r_i(j)$, in Equation (5.4) as the difference of miss rate and invalidation rate (buffer purge rate).

$$r_i(j) = \lambda N_L \alpha_i (1 - \frac{Y_i(j)}{\beta_i L}) - (N-1)\lambda N_L \alpha_i p_i^u \frac{Y_i(j)}{\beta_i L}. \tag{5.8}$$

(Note that there are no invalid granules under IN.) Together with Equations (5.6) and (5.3), it can be solved iteratively with the initial condition of $p_i(1) = \alpha_i$. Hence,

$$H^{bi} = \sum_{i=1}^{M} \alpha_i Y_i(B)/(\beta_i L). \tag{5.9}$$

It was found in [DAN93B] that the difference in buffer hit probabilities between CA and IN to be very sensitive to the data access pattern. For skewed data accesses, the difference tends to be small, as under IN the immediate identification of invalidated granules for buffer reuse results mainly in more cold granules being buffered. In the presence of invalidation effect, as the buffer increases, a saturation point on buffer hit probability will be reached even under IN. This occurs at the point where the replacement rate is zero, i.e., at a buffer size of $L/(1+(N-1)p^u)$ as can be derived from (5.8). Furthermore, from (5.8), it can be shown that the buffer hit probability at this point is roughly $1/(1+(N-1)p^u)$ for each partition, cold or hot. This upper limit on buffer hit probability is independent of the access pattern, uniform or skew. This is in contrast to the single node case where with sufficient buffer, the buffer hit probability can go to one.

6. Extensions on Buffer Modeling

6.1 Lock Retention, Deferred Writes, and Remote Caching

The buffer modelling methodology can be extended to study various other buffer related issues. In a single node environment, deferred write policy, where the dirty granules are not forced to disk at commit time, is often used to save write IO's as in IBM DB2. The number of write IO's is reduced if a granule can get multiple updates from different transactions before it is flushed out from the buffer and written to disks. The write IO rate under the deferred write policy can be determined by the rate that dirty granules are flushed out from the buffer, i.e. the dirty fraction of $\sum_{i=1}^{M} r_i^u(B)$. The write I/O reduction can be analyzed by extending the above methodology to distinguish between clean and dirty granules [DAN93A].

The two principles on conservation of flow and relative push down rate still hold for the dirty (respectively, the clean) granules of each partition. However, there is an additional complexity. Consider the CA (similarly for the IN) scheme. In the previous section, Equations (5.2)-(5.7) form a set of forward recurrence relations. This is due to the fact that in Equation

(5.5) for the conservation of flow, only the content in the top stack j positions matters. This is referred to as the substack decomposable property in [DAN92B]. However, if we apply the conservation of flow to the dirty pages of partition i, the first term on the right hand side of Equation (5.5) is no longer the miss probability at the top j stack positions. Instead, it is now the probability that the remaining stack positions (from location $j+1$ to B) containing the dirty granule requested. (In the previous section, we only need to know that the requested granule is missing from the top substack. Now the fact that the requested granule is missing from the top substack does not imply that a dirty granule would be brought to the top of the stack. This can only occur if the requested granule is already present in the bottom portion of the buffer and is dirty.) We thus lose the substack decomposable property. This makes the recurrence relations more complex to solve.

In a multi-node environment, the delaying propagation of dirty granules to disks or server nodes can cause coherency problem for other nodes trying to access these dirty granules. This problem can be solved via **lock retention** on dirty granules where locks are continued to be held after commit by the buffering node as long as the granules are in the buffer. The lock manager can thus identify the node owning the most up-to-date version of a granule and properly direct a requesting node where to obtain the up-to-date version of a granule. With the ever increasing gap between memory and disk access times, the concept of remote caching has been pursued by various researchers [LEFF91, LI89, FRAN92, MOHA91]. Under remote caching, granules can be directly transferred between buffers. (In essence, the buffers of all the other nodes become another level of the storage hierarchy between local buffer and disks.) Note that although delaying propagation of dirty granules to disks can improve normal performance, it can prolong the recovery time upon system failure as all dirty granules need to be derived from the database log and applied to the database. For the multi-node environment, the recovery process can be further complicated as log entries from all nodes need to be merged and applied in order, if dirty granules can be transferred to multiple nodes via the remote caching mechanism before writing to disks [MOHA91].

Generalizations of the buffer coherency schemes to support lock retention, remote caching and delayed dirty page propagations have been proposed in [MOHA91, DAN92A,93A] for a data-sharing like environment and for a client-server like environment in [FRAN92]. The buffer analysis presented here can be extended to study the performance improvement of these schemes [DAN92A, DAN93A] and their recovery time trade-offs [DAN93B]. The extension needs to track the buffer composition in terms of the lock mode held in each buffer location, in addition to the partition type of a granule. The two principles on conservation of flow and relative push down rate still hold for granules of each lock mode held, but again we lose the substack decomposable property.

6.2 Shared Buffer

As pointed out in the previous section, in a multi-node environment, the cross invalidation effect puts a upper limit on the attainable buffer hit probability and the usable amount of local buffer. Furthermore, granule replications among the local buffers also reduce the effectiveness of buffer usage as compared to a single node environment. Shared buffer has been considered as a means of providing more effective buffering, e.g. [DAN91, YU92B,C, FRAN92]. In [DAN91], a unified analytical modelling methodology is developed to study various shared buffer placement polices. These polices differ on the types of granules selected to place in the shared buffer, such as updated granules, missed granules, replaced granules from the local buffer, and any combinations of these three types of granules. Note that in the shared buffer, only the granules that are not already duplicated in the local buffer can help improve the overall buffer hit probability. The analysis in [DAN91] captures the dependency between local and shared buffer contents under each policy.

7. Summary and Conclusion

In transaction processing, the complexity of the system makes it difficult to understand the trade-offs between the various design alternatives without detailed analysis. Simulation models of transaction processing systems can often be very time consuming to run. Therefore, for exploring a wide variety of design alternatives it would be useful to have a unified analytical methodology. Furthermore, from simulation studies, it is generally hard to pinpoint the causality of the results, extrapolate the findings from the environment studied to other environments, and reconcile the differences among other related studies, as the large number of parameters can interact in very complex ways. On the other hand, an analytic expression can provide far more insight into the causality effect of the various parameters, and be used by other practitioners to study different environments under another sets of parameter values.

In this paper we reported some recent advancement in analytical modelling methodology for transaction processing systems. A unified approximate analytical methodology is reported to study various aspects of the transaction processing systems, including the effects of different CC schemes and coherency control schemes over a spectrum of environments including single node, to multi-node cluster or client-server environment. The methodology presented decomposes the model into submodels for hardware resource contention, for data contention and for buffer hit. We focus on the data contention model and buffer model. The data contention model is applicable to various CC schemes, including locking, different OCC and hybrid schemes. The buffer model can be applied to various buffer coherency schemes, including check on access scheme, broadcast or selective notification scheme, and other extensions with lock retention to support remote caching and delayed dirty page propagations.

References

[BERN87] Bernstein, P.A., Hadzilacos, V., and Goodman, N., "Concurrency Control and Recovery in Database Systems", Addison Wesley, 1987.

[BAYE82] Bayer, R., et al., "Dynamic Timestamp Allocation for Transactions in Database Systems", In H.-J. Schneider, editor, Proc. of 2nd Intl. Symp. on Distributed Data Bases, pp. 9-21, North Holland, 1982.

[BOKS87] Boksenbaum, C., et al., "Concurrent Certifications by Intervals of Timestamps in Distributed Database Systems", IEEE Transactions on Software Engineering, Vol. SE-13, No. 4, April 1987, pp. 409-419.

[CARE91] Carey, M.J., et al., "Data Caching Tradeoffs in Client-Server DBMS Architectures", ACM SIGMOD, Denver, CO, May 1991, pp. 357-366.

[CELL88] Cellary, W., Gelenbe, E., and Tadeusz, M., "Concurrency Control in Distributed Database Systems", North-Holland, 1988.

[CHES83] Chesnais, A., Gelenbe, E., and Mitrani, I., "On the Modeling of Parallel Access to Shared Data", Comm. ACM, Vol. 26, No. 3., Mar. 1983, pp. 196-202.

[CICI90A] Ciciani, B., Dias, D.M., and Yu, P.S. "Analysis of Replication in Distributed Database Systems", IEEE Transactions on Knowledge and Data Engineering, Vol. 2, No. 2, June 1990, pp. 247-261.

[CICI90B] Ciciani, B., Dias, D.M., Iyer, B.R., and Yu, P.S., "A Hybrid Distributed Centralized System Structure for Transaction Processing", IEEE Transactions on Software Engineering, Vol. 16, No. 8, Aug. 1990, pp. 791-806.

[CICI92] Ciciani, B., Dias, D.M., and Yu, P.S. "Analysis of Concurrency-Coherency Control Protocols for Distributed Transaction Processing with Regional Locality", IEEE Transactions on Software Engineering, Vol. 18, No. 10, Oct. 1992, pp. 899-914.

[CORN86] Cornell, D.W., Dias, D.M. and Yu, P.S., "On Multisystem Coupling Through Function Request Shipping", IEEE Transactions on Software Engineering, Vol. SE-12, No.10, October 1986, pp. 1006-1017.

[DATE83] Date, C.J., "An Introduction to Database Systems", Vol. 2, Addison Wesley, Reading, MA, 1983.

[DAN88] Dan, A., Towsley, D.F., and Kohler, W.H., "Modeling the Effects of Data and Resource Contention on the Performance of Optimistic Concurrency Control Protocols", Proc. 4th Intl. Conf. on Data Engineering, Los Angeles, CA, Feb. 1988, pp. 418-425.

[DAN90A] Dan, A., and Towsley, D., "An Approximate Analysis of the LRU and FIFO Buffer Replacement Schemes", ACM SIGMETRICS, Denver, CO, (Performance Evaluation Review, Vol. 18, No. 1), May 1990, pp. 143-152.

[DAN90B] Dan, A., Dias, D. M. and Yu, P. S., "The Effect of Skewed Data Access on Buffer Hits and Data Contention in a Data Sharing Environment", Proc. 16th Intl. Conf. on Very Large Databases, Brisbane, Australia, Aug. 1990, pp. 419-431.

[DAN91] Dan, A., Dias, D.M., and Yu, P.S., "Analytic Modelling of a Hierarchical Buffer for a Data Sharing Environment", Proc. 1991 ACM SIGMETRICS Conference, San Diego, CA, May 1991, pp. 156-167.

[DAN92A] Dan, A., and Yu, P.S., "Performance analysis of coherency control policies through lock retention", Proc. of the ACM SIGMOD Intl. Conf. on Management of Data, San Diego, CA, June 1992, pp. 114-123.

[DAN92B] Dan, A., Yu, P.S. and Jhingran, A. "Recovery Analysis of Data Sharing Systems under Deferred Dirty Page Propagation Policies", IBM research Report RC 18553, Yorktown Heights, 1992.

[DAN93A] Dan, A. and Yu, P.S., "Analytic Modeling and Comparison of Buffer Coherency Policies based on Lock Retention", IBM research Report RC 18664, Yorktown Heights, 1993.

[DAN93B] Dan, A., and Yu, P.S., "Performance Analysis of Buffer Coherency Policies in a Multi-System Data Sharing Environment," IEEE Transactions on Parallel and Distributed Systems, Vol. 4, No. 3, March 1993.

[DAN93C] Dan, A., Yu, P.S., and Chung, J-Y., "Database Access Characterization for Buffer Hit Prediction", Proc. 9th Intl. Conf. on Data Engineering, Vienna, Austria, April 1993.

[DIAS88] Dias, D. M., Iyer, B. R. and Yu, P. S., "Trade-offs Between Coupling Small and Large Processors for Transaction Processing", IEEE Transactions on Computers, Vol. 37, No. 3, March 1988, pp. 310-320.

[DEUX90] Deux, O., et al, "The Story of O_2", IEEE Transactions on Knowledge and Data Engineering, Vol. 2, No. 1, March, 1990, pp. 91-108.

[FRA85] Franaszek, P.A. and Robinson, J.T., "Limitations of Concurrency in Transaction Processing", ACM Transactions on Database Systems, Vol. 10, No. 1, March 1985, pp. 1-28.

[FRA92] Franaszek, P.A., Robinson, J.T. and Thomasian, A., "Concurrency Control for High Contention Environments", ACM Transactions on Database Systems, Vol. 17, No. 2, June 1992, pp. 304-345.

[FRAN92] Franklin, M.J., and Carey, M.J. and Livny, M., "Global Memory Management in Client Server DBMS Architectures", Proc. 18th Intl. Conf. on Very Large Databases, Vancouver, Canada, Aug. 1992, pp. 596-609.

[GRAY81] Gray, J., Homan, P., Obermarck, R. and Korth, H., "A Straw Man Analysis of Probability of Waiting and Deadlock", IBM Research Report RJ 3066, San Jose, CA, 1981.

[GRAY86] Gray, J.N., "An Approach to Decentralized Computer Systems", IEEE Transactions on Software Engineering, Vol SE-12, No. 6, June 1986, pp. 684-692.

[HART89] Hartzman, C. S., "The Delay Due to Dynamic Two-Phase Locking", IEEE Transactions on Software Engineering, Vol. 15, No. 1, Jan. 1989, pp. 72-82.

[HORN87] Hornick, M., and Zdonik, S., "A Shared, Segmented Memory System for an Object-Oriented Database", ACM Transactions on Information Systems, Vol. 5, No. 1, Jan, 1987.

[HSU88] Hsu, M., and Shang, B., "Modeling Performance Impact of Hotspots", Technical Report TR-08-88, Aiken Computation Lab., Harvard University, April 1988.

[HSU92] Hsu, M., and Zhang, B., "Performance Evaluation of Cautious Waiting", ACM Transactions on Database Systems, Vol. 17, No. 3, Sept. 1992. pp. 477-512.

[KIM90] Kim, W., et al, "The Architecture of the ORION Next Generation Database System", IEEE Transactions on Knowledge and Data Engineering, Vol. 2, No. 1, March 1990, pp. 109-124.

[KRON86] Kronenberg, N., Levy, H., and Strecker, W.D., "VAXcluster: a Closely-Coupled Distributed System", ACM Transactions on Computer Systems, Vol. 4, No. 2, May 1986, pp. 130-146.

[KUNG81] Kung, H.T. and Robinson, J.T., "On Optimistic Methods for Concurrency Control", ACM Transactions on Database Systems, Vol. 6, No. 2, June 1981, pp. 213-226.

[LAVE83] Lavenberg, S.S. (Ed.), "Computer Performance Modeling Handbook", Academic Press, 1983.

[LEFF91] Leff, A., Yu, P.S., and Wolf, J.L, "Policies for Efficient Memory Utilization in a Remote Caching Architecture", Proc. 1st Intl. Conf. on Parallel and Distributed Information Systems, Miami Beach, Florida, Dec. 1992, pp. 198-205.

[LI89] Li, K., and Hudak, P., "Memory Coherence in Shared Virtual Memory Systems," ACM Transactions on Computer System, Vol. 7, Nov. 1989, pp. 321-359.

[MOHA91] Mohan, C., and Narang, I., "Recovery and Coherency Control Protocols for Fast Intersystem Page Transfer and Fine Granularity Locking in a Shared Disks Transaction Environment", Proc. 17th Intl. Conf. on Very Large Databases, Barcelona, Spain, Sept. 1991, pp. 193-207.

[MORR85] Morris, R.J.T., Wong, W.S., "Performance Analysis of Locking and Optimistic Concurrency Control Algorithms", Performance Evaluation, Vol. 5, 1985, pp. 105-118.

[RAHM86] Rahm, E., "Primary Copy Synchronization for DB-Sharing", Information Systems, Vol. 11, No. 4, 1986, pp. 275-286.

[RAHM88] Rahm, E., "Emprical Performance Evaluation of Concurrency and Coherency Control Protocols for Data Sharing", IBM Research Report RC 14325, 1988.

[RAMA89] Ramachandran, U., Ahamad, M. and Khalidi, M.Y.A., "Coherence of Distributed Shared Memory: Unifying Synchronization and Data Transfer", Proc. 18th Intl. Conf. on Parallel Processing, St. Charles, Ill, Aug. 1989, pp. II-160--II-169.

[RYU87] Ryu, I. K. and Thomasian, A., "Performance Analysis of Centralized Databases with Optimistic Concurrency Control", Performance Evaluation, Vol. 7, 1987, pp. 195-211.

[ROSE78] Rosenkrantz, D.J., Stearns, R.E., and Lewis, P.M., II. "System Level Concurrency Control for Distributed Database Systems", ACM Transactions on Database Systems, Vol. 3, No. 2, June 1978, pp. 178-198.

[SEVC83] Sevcik, K. C., "Comparison of Concurrency Control Methods Using Analytic Models", Information Processing 83, R.E.A. Mason (ed.), North Holland, 1983, pp. 847-858.

[SING88] Singhal, M. and Yesha, Y., "A Polynomial Algorithm for Computation of the Probability of Conflicts in a Database under Arbitrary Data Access Distribution", Information Processing Letters, Vol. 27, No., 2, Feb. 1988, pp. 69-74.

[SING91] Singhal, M., "Analysis of the Probability of Transaction Abort and Throughput of Two Timestamp Ordering Algorithms for Database Systems", IEEE Transaction on Knowledge and Data Engineering, Vo. 3, No. 2, June 1991, pp. 261-266.

[TAY84] Tay, Y.C., "A Mean Value Performance Model for Locking in Databases", Ph.D. Dissertation, Harvard University, Cambridge, MA, Feb. 1984.

[TAY85A] Tay, Y.C., Suri, R. and Goodman, N., "A Mean Value Performance Model for Locking in Databases: The No-Waiting Case", Journal of the ACM, Vol. 32, No. 3, July 1985, pp. 618-651.

[TAY85B] Tay, Y.C., Goodman, N., and Suri, R., "Locking Performance in Centralized Databases", ACM Transactions on Database Systems, Vol. 10, No. 4, Dec. 1985, pp. 415-462.

[TAY90] Tay, Y.C., "Issues in Modelling Locking Performance", in "Stochastic Analysis of Computer and Communication Systems, H. Takagi (Ed.), North-Holland, 1990, pp. 631-655.

[THOM85] Thomasian, A. and Ryu, I.K., "Analysis of Some Optimistic Concurrency Control Schemes Based on Certification", Performance Eval. Review, 13, 2 (Proc. of 1985 ACM SIGMETRICS), pp.192-203.

[THOM91] Thomasian, A. and Ryu, I.K., "Performance Analysis of Two-Phase Locking", IEEE Transactions on Software Engineering, Vol. 17, No. 5, May 1991, pp. 386-401.

[WILK90] Wilkinson, K., and Neimat, M.A., "Maintaining Consistency of Client-Cached Data" Proc. 16th Very Large Database Conf., Brisbane, Australia, August 1990, pp. 122-133.

[YU85] Yu, P.S., Dias, D.M., Robinson, J.T., Iyer, B.R. and Cornell, D.W., "Modelling of Centralized Concurrency Control in Multi-System Environment", Performance Eval. Review, 13, 2 (Proc. of 1985 ACM SIGMETRICS), pp.183-191.

[YU87] Yu, P.S., et al., "On Coupling Multi-Systems Through Data Sharing", Proceedings of the IEEE, Vol. 75, No. 5, May 1987, pp. 573-587.

[YU90] Yu, P.S. and Dias, D.M., "Concurrency Control Using Locking with Deferred Blocking", Proc. 6th Intl. Conf. on Data Engineering, Los Angeles, CA, 1990, pp.30-36.

[YU91] Yu, P.S., Heiss, H. and Dias, D.M., "Modelling and Analysis of a Time-Stamp History Based Certification Protocol for Concurrency Control", IEEE Transactions on Knowledge and Data Engineering, Vo. 3, No. 4, Dec. 1991, pp. 525-537.

[YU92A] Yu, P.S. and Dias, D.M., "Analysis of Hybrid Concurrency Control Schemes for a High Data Contention Environment", IEEE Transactions on Software Engineering, Vol. 18, No. 2, Feb. 1992, pp. 118-129.

[YU92B] Yu, P.S. and Dan, A., "Effect of System Dynamics on Coupling Architectures for Transaction Processing", Proc. 8th Intl. Conf. on Data Engineering, Tempe, AZ, Feb. 1992, pp. 458-469.

[YU92C] Yu, P.S., and Dan, A., "Impact of Workload Partitionability on the Performance of Coupling Architectures for Transaction Processing", Proc. 4th IEEE Symposium on Parallel and Distributed Processing, Dec. 1992, pp. 40-49.

[YU93] Yu, P.S., Dias, D., and Lavenberg, S.S., "On the Analytical Modeling of Database Concurrency Control", Journal of the ACM, Sept. 1993.

[YU94] Yu, P.S., K.L. Wu, K.J. Lin, and S.H. Son, "On Real-time Databases: Concurrency Control and Scheduling", (to appear) Proceedings of the IEEE, Jan. 1994.