Simulation studies on concurrency control in parallel transaction processing systems

Jiahong Wang *, Jie Li 1, Hisao Kameda 2

Institute of Information Sciences and Electronics, University of Tsukuba, Tsukuba Science City, Ibaraki 305, Japan

Received 29 May 1996; accepted 25 October 1996

Abstract

Parallel transaction processing (TP) systems have great potential to serve the ever-increasing demands for high transaction processing rate. This potential, however, may not be reached due to the concurrency control (CC) requirements. In this paper a simulation system for shared-nothing parallel TP systems was presented, which aims at, but is not restricted to, the studies of the CC methods. A distributed locking-based CC method called LW DC_k (local wait-depth control) was also proposed for overcoming the drawbacks of the widely-used two-phase locking (2PL) CC method. LW DC_k was compared with 2PL and the well-known DWDL (distributed wait-depth limited) CC methods based on the presented simulation system. The application of this simulation system demonstrated its effectiveness. Simulation studies indicated also that the proposed method outperforms 2PL and DWDL.

Keywords: Concurrency control; Parallel algorithms; Parallel database; Simulation model; Two-phase locking

1. Introduction

To meet the ever-increasing demands for high transaction processing rate, so-called shared-nothing parallel transaction processing (TP) architecture has been proposed [23]. In such architecture, there are numerous processors connected by an interconnection network; each of these processors accesses its own disks and memory; each relation of
database is partitioned among multiple network nodes; the transaction execution is
distributed over the network. The concept of shared-nothing has been adopted by
numerous well-known parallel TP systems such as Tandem [24], Teradata [26], Gamma
[11], and Bubba [7]. The shared-nothing parallel TP systems are becoming more and
more popular for their cost effectiveness, scalability, and availability. Their perfor-
ance, however, may be restricted due to the requirements of the concurrency control
(CC), and so does the degree of system parallelism [16]. We, therefore, are interested in
performance studies of CC methods for shared-nothing parallel TP systems.

Concurrency control is one of the most critical performance factors in parallel TP
systems. So far there have been several simulation models (e.g., [5,10,16,19]) that can be
used directly or indirectly to study CC methods for shared-nothing parallel TP systems.
However, it seems that few of them can do that perfectly. Some models such as those
presented in [5,10] are too simple to catch the details of the real-life TP systems. With
these models only such a transaction that consists of several independent subtransactions
can be processed. That is, it is assumed that a transaction is divided into several
subtransactions, and there is no communication between any two of them. This is an
unrealistic assumption, and most of the real-life transactions, such as those arising in
TPC-C [29] and Wisconsin [15] benchmarks, have more complex structure.

Some other models such as those presented in [16,19] reflect the real-life parallel TP
environment more properly than those stated above. Unfortunately, workloads used by
them are unsuitable for evaluating CC methods. In fact, we found that most of the
existing models for parallel TP systems have the similar problem. In [16,19] database
benchmarks and real-life transactions are used as their workloads. We think database
benchmarks and real-life transactions are suitable for evaluating TP systems since a
given TP system is oriented to a specified application environment. For the CC methods,
however, it is doubtful since a CC method is generally required to be applicable to a
variety of application environments. As an example, if we use a query-bound benchmark
to compare two well-known CC methods: the two-phase locking (2PL) [13] and the
optimistic control CC [17] methods, we would be misguided into thinking that 2PL is
inferior to the optimistic control CC method, which is not true in most of the cases
according to the past studies. Another example is that the real-life transactions used in
[16] led to such an uncompleted conclusion that a large degree of system parallelism is
beneficial [7].

In addition, most previous studies emphasized the development of new algorithms or
the comparison of different algorithms rather than providing accurate performance
model. All their works were done by using simple performance models oriented to their
particular problems. We distinguish our work by presenting a comprehensive simulation
system for shared-nothing parallel TP systems. There are two distinctive features for our
simulation system. The one is that it simulates the dataflow-oriented parallel TP systems
such as Bubba and Gamma by integrating the mature simulation techniques presented in
[1,5,8]. The other is that it introduces a workload model which is oriented to the studies
of CC methods. In addition to CC methods, other performance factors of shared-nothing
parallel TP systems can also be studied by using this simulation system, some examples
of which are the degree of system parallelism, the level of data and resource contention,
the CPU/Disks scheduling strategy, the deadlock detection/resolution method, and the
transaction restart policy. This simulation system enables users to study the performance behavior of the factors given above under diverse environments to provide insights into different strategies.

Another contribution of this paper is a new restart-oriented locking-based CC method called LW DC$_k$ (local wait-depth control), which is proposed for overcoming the drawbacks of 2PL for shared-nothing parallel TP systems. Like centralized TP systems, most of the shared-nothing parallel TP systems use 2PL to resolve data contention so as to maintain data consistency and integrity. As the level of the data contention increases (for example, by increasing the multiprogramming level (MPL), i.e., the total number of transactions running concurrently in the system), however, the interaction between data contention and 2PL tends to cause system performance degradation due to transaction blocking and aborting. As an extreme of this negative effect, data contention thrashing may occur [27]. Data contention thrashing is such a phenomenon that as MPL increases, system throughput grows almost linearly first, flattens out then, and then drops suddenly. Such drawbacks of 2PL can be attributed to the fact that blocked transactions hold locks that they have acquired, and result in further blocking (a snowball effect) [27], and can be overcome by LW DC$_k$.

LW DC$_k$ judiciously selects some transactions to restart so as to limit the wait-depth of the blocked transactions (i.e., the number of the transactions for which a blocked transaction has to wait before it can proceed again). Numerous 2PL performance studies (e.g., [28]) indicated that unlimited waiting degrades system performance. Therefore, TP systems with LW DC$_k$ would work well in a wide spectrum of data contention. Another important property of LW DC$_k$ is that it limits the wait-depth of blocked transactions locally. That is, the wait-depth of blocked transactions is limited without negotiation across the network, although the lock conflicts causing a wait-chain may occur at more than one node. In addition, LW DC$_k$ can prevent the occurrence of deadlocks.

Among the approaches proposed in the past with the same purpose as ours is the well-known one called DWDL (distributed wait-depth limited) [14]. It overcomes the drawbacks of 2PL by limiting the wait-depth of blocked transactions to one by restarting some transactions whenever wait-chains longer than one occur. The lock conflict resolution mechanism of DWDL demands some centralized control nodes, i.e., DWDL is a centralized approach. Each time a lock conflict occurs at a local node, it is necessary to inform the centralized control nodes of this lock conflict event. For DWDL both the communication overhead and the restart rate are very high, which would degrade system performance.

As an application example, the proposed simulation system is used to evaluate LW DC$_k$, 2PL, and DWDL CC methods. Simulation experiments indicate that this simulation system works well in evaluating CC methods. In addition, the simulation results show that the proposed CC method offers better system performance than 2PL and DWDL.

This paper is organized as follows. In Section 2 we present our simulation system. In Section 3 we describe 2PL, DWDL, and the proposed LW DC$_k$ CC methods. In Section 4 an application example of our simulation system is given by evaluating 2PL, DWDL, and the proposed LW DC$_k$. Simulation results and discussion are also given in Section 4. Conclusions appear in Section 5.
2. Simulating shared-nothing parallel TP systems

In this section we present our simulation system by describing its overall structure first, workload model then, and then the simulation model. The object system is shown in Fig. 1. Transactions arriving at the system are accepted and started at front-end nodes. The final results are also routed by these front-end nodes to their users. The actual transaction processing takes place at the nodes identified by PE. Transactions are executed according to dataflow control strategy [3], the work is done where the data needed reside.

2.1. Overall structure of our simulation system

The overall structure of our simulation system is shown in Fig. 2. It consists of three parts: specifications, algorithms, and simulation model.

All the specifications are included in a parameter file. We can modify this file to meet different requirements. A simplified example of these specifications can be found in Section 4. Transaction generation and CC algorithms are implemented in two separate modules so that the new algorithms can be added easily. The simulation model is a closed model which simulates the dataflow-oriented shared-nothing parallel TP systems similar to Bubba [7] and Gamma [12] by integrating the mature simulation techniques presented in [1,5,8]. This model has been implemented as a simulation program by using C programming language and the SMPL software library [18]. The performance evaluation system describes the exact system to be examined as well as the simulation experiments to be executed.

2.2. Workload model

A widely accepted method for evaluating CC methods in centralized TP systems is to model a transaction as a sequence of database actions. Each action involves a lock request for the granule to be accessed, followed by the granule access to disk, and followed by a period of CPU usage for processing this granule. How to extend such a workload model to the parallel case is a challenge. For the simulation system of parallel TP systems with the aim of evaluating CC methods, the workload model should consider the following factors:

![Fig. 1. Shared-nothing parallel TP systems.](image-url)
(1) **Intra-transaction Parallelism.** In addition to the concurrent execution of multiple transactions, in parallel TP systems there also exists another kind of parallelism, i.e., the intra-transaction parallelism. If a transaction is viewed as a collection of SQL statements, then the intra-transaction parallelism includes (1) the concurrent execution of multiple SQL statements in the same transaction; (2) parallel processing of multiple operators in the same SQL statement; (3) parallel processing of the same operator in an SQL statement. The workload selected should facilitate the specification of the intra-transaction parallelism. An inadequate view of transactions cannot reveal such a parallelism.

(2) **Intra-transaction communication overhead.** In parallel TP systems, the communication overhead within a transaction execution needs to be considered. Too simple a view of transactions tends to cause underestimates of this overhead. The workload selected should facilitate the estimate of the communication overhead within a transaction execution.

(3) **Data and resource contention.** In TP systems there are two kinds of contention: data contention and resource contention. Either of them affects system performance. Timing of all the events involved is crucial in evaluating the effect of these two kinds of contention on system performance. The view of transactions should be complex enough to provide realistic estimate of the time when a distinct event occurs.

(4) **Flexibility and variety.** Considering that a CC method may be used in a variety of TP systems, the workload used to evaluate CC methods should be flexible enough to model most application environments, and should be able to provide all the possible transaction patterns.

(5) **Standard.** A problem found in most previous simulation models is the lack of consistency in their approaches. This makes it impossible to compare results among researchers. The standard approach must be based on a unified transaction representation. We think this representation should be generalized without characteristics of applications, since the same application implemented on different systems would present different workloads to these systems [2].

On the basis of the above observations, we use a so-called **transaction dataflow graph** (i.e., the transaction execution plan generated during the compilation of transac-
tion programs in real-life TP systems) to represent transactions. In fact, graph models have been widely used to study the behavior of concurrent systems [22]. Transaction dataflow graph is a directed acyclic graph, where graph nodes represent transaction fragments called components and arcs the data dependencies between the components. Data are viewed as flowing through the arcs from one component to another in a stream of discrete tokens. The scheduling of components is constrained only by the data dependencies identified by the graph. Components in the same directed path must be scheduled serially, and components in different paths can be scheduled independently. A component accesses at most one relation of database. A component that accesses a relation consists of a collection of read (or update) actions that act on this relation. Each action involves a lock request for the granule to be accessed, followed by the granule access to disk, and followed by a period of CPU usage for processing this granule.

We found that all the requirements stated above can be satisfied by our workload
The intra-transaction parallelism can be implemented by processing the same and the different components in parallel. The intra-transaction communication overhead can be estimated and all the necessary events can be timed, since the decisions concerning distributed execution of a transaction on the shared-nothing architecture can be explicitly expressed. This workload model is flexible and variable due to the properties of the graph representation. Finally, this workload model can be expected to be standard since it is general and corresponds to a unified transaction representation.

An algorithm for generating dataflow graphs with arbitrary topologies has been designed. As an example, relational algebra query trees [6] have been used for our simulation experiments in Section 4. Each relational algebra query tree comprises one or more operators and is organized in tree form (Fig. 3). Nodes in the tree correspond to the relational algebra operators. Leaf nodes operate on the relations of the database. Nonleaf nodes operate on the temporary relations that are produced by nodes immediately below them. The results are given at the root node.

TPC-A [15] and a subset of Wisconsin benchmark [15] were also represented by dataflow graphs so as to check the capacity of our workload model. Another more complex example of transaction dataflow graph will be given in Fig. 5 in Section 2.3.

Transactions are classified. Transactions in the same class have the same characteristics, such as dataflow graph, occurrence frequency, and the number of granules required. In addition, in TP systems relations of database and records in a relation may not be equally likely accessed. We model such a kind of access skew by the well-known b–e rule. That is, b% of the accesses goes to e% of the data (relations or records).

2.3. Simulation model

The simulation model is shown in Fig. 4. The structure of this simulation model, the relationship of this simulation model and the workload model, and the details of workload processing are given in Fig. 5. Note that although the operations such as sending and receiving messages consume CPU time, they are not depicted in these two figures for the sake of brevity.
2.3.1. The computer system

The computer system consists of an arbitrary number of nodes (called processing element, or PE) connected by an interconnection network. Each PE has its own processor, memory, disks, and communication processor. Each PE also has a copy of the operating system that supports a multi-threaded process environment. Front-end nodes are configured for interfacing with users and managing the system. A database cache is provided for each PE for caching local data. Disks have been explicitly modeled as servers to capture I/O bottlenecks.

2.3.2. The database

The database is a collection of relations, and a relation is a collection of records. One or more records (or pages) constitute a database granule, which is a lockable unit. All granules have the same size. The size of the granule can be changed by changing the number of records (or pages) per granule. Each relation is horizontally partitioned among multiple PEs. Database partitions can be kept memory-resident all or partially. Data replication is not considered here because it is less desirable in the shared-nothing environment.

Database is represented by a database description file in our simulation system. For each relation in database, database description file provides such information as relation name, relation size by the record, record size by the byte, total number of partitions included, and the size and site of each partition. Data are described and allocated to every PE by editing system parameter file manually, and then generating the database description file automatically. Alternatively, the database description file can be edited manually so that different data allocation strategies can be adopted easily.

2.3.3. The transaction processing

Transactions are processed according to dataflow control strategy [3]. When a transaction arrives at the system, it is accepted and preprocessed by a transaction manager that resides at a front-end node. Transaction managers are designated to handle all the management functions, such as transaction initiation, commit, and restart. Transaction manager starts a transaction by first loading all its component codes into their corresponding PEs where the data needed reside, and then initiating all the ancestor components, i.e., those components that have no data dependency on any others. It is said that a component has data dependency on another component if it uses results of that component.

As stated in Section 2.2, a component is no more than a transaction fragment. Since the relation accessed by a component is partitioned among multiple PEs, each component is implemented by assigning a group of parallel threads called sibling threads to it, one per PE where the data needed reside. Each sibling thread executes the same component code copy, and passes its intermediate results to its successors. Consequently, the transaction execution is distributed over the network, and the work is done where the data needed reside. Note that if a component accesses no relations, its execution sites follow one of the predecessors of this component.

Since the dataflow control strategy is adopted, only the ancestor components need to be initiated explicitly, all the others are activated and synchronized entirely by dataflows.
From the viewpoint of implementation, each sibling thread of a non-ancestor component is preloaded into the sibling-thread pool at the corresponding PE to wait for the required data. When a component produced its results and these results are to be used by some other components, they are sent to the corresponding PEs. The intermediate results received from the network are directed to the data pool, which then can be used by the corresponding thread. A sibling thread of non-ancestor components is dynamically activated when the first data message addressed to it arrives. It is assumed that each thread sends its intermediate results to all of its successor threads.

For a sibling thread of a component that accesses a relation of database, right after initially initiated (if this component is an ancestor component) or for each data message received (if this component is a non-ancestor component), this sibling thread cycles through CPU and disk queues several times. Before a granule is accessed, CC is required. Depending on the result of CC, one of the following three is chosen: this thread proceeds to access disks; this thread is entered into a blocked queue to wait until it can proceed again; some transactions are aborted. See Section 3 for details of different CC methods. If a transaction is found having completed all its data requests (by detecting EOT), a two-phase commit protocol is started. The modified data are forced into stable storage as a part of commit processing. The committed transaction is immediately replaced by a new one.

Intermediate results are produced in each cycle. These results are sent out each time they can contribute to a network packet. For a thread of ancestor components, when completing all its data requests, it terminates. For a thread of non-ancestor components, after processing all the data messages in the data pool, depending on whether all its predecessors have completed, it either remains in sibling-thread pool, or exits from it and terminates.

Deadlock is possible for TP systems. In our simulation system, local deadlocks are detected immediately based on a local wait-for-graph whenever a lock conflict occurs. Global deadlocks, i.e., deadlocks across several PEs, are resolved by periodically collecting local wait-for information from all PEs by a centralized system component that resides at a specified front-end node. In resolving a deadlock, the youngest transaction among those involved in the deadlock is aborted and to be restarted. The aborted transaction waits for its blocker to be committed or aborted. The restarted transaction repeats its lock requests as those before it was aborted. Alternatively, the aborted transaction can be allowed to re-enter the system after an exponentially distributed delay time, and the restarted transaction resamples the locks it needs [25]. The latter is what we adopted in our simulation experiment in Section 4.

3. Concurrency control methods

In this section we describe three CC methods: 2PL, DWDL, and the proposed LW DC_k. These CC methods will be evaluated in Section 4 as an application example of our simulation system. For DWDL and LW DC_k, it is assumed that we are concerned with only such transactions as those arising in business applications which have stringent response time requirements and do not interact with users by themselves. Note that in
our simulation system, a transaction is processed by dividing this transaction into several components each of which consists of a group of sibling threads.

3.1. The two-phase locking (2PL) method

This is the commonly used method in distributed TP systems. Transactions set locks directly at the primary execution node and indirectly through their subtransactions at other nodes. All locks are held until a transaction is either successfully committed or aborted [4]. Lock conflicts between transactions are resolved by making the lock requester wait for the lock holder in an FCFS queue; these waiting-for relationships form a wait-for graph (WFG). A centralized deadlock detection scheme as what we use (see Section 2.3.3) may be adopted for deadlock detection. Alternatively, distributed detection schemes and the schemes based on timeout mechanism may also be used.

3.2. The distributed wait-depth limited (DWDL) method

Nowadays for overcoming the drawbacks of 2PL, several approaches have been proposed (e.g., [9,20,28]). DWDL [14] is a well-known one among them. DWDL is a restart-oriented locking-based CC method. That is, it schedules transactions by means of locking, and may restart some transactions in resolving lock conflicts. By contrast, 2PL is a waiting-oriented CC method which is inclined to block the transaction that encounters a lock conflict.

DWDL limits the wait-depth of blocked transactions to one by restarting some transactions whenever wait-chains longer than one occur. In DWDL an age function is used to reflect the progress of transactions. Given a transaction \( T \), its age is defined to be the difference of the current time and its starting time (for its latest invocation if \( T \) is restarted), denoted by \( L(T) \). One of the four restart rules of DWDL is such that \( T_1 \) in the wait-chain of \( T_0 \rightarrow T_1 \rightarrow T_2 \) is restarted unless its age is larger than those of \( T_0 \) and \( T_2 \), in which case \( T_0 \) is restarted. The other three rules are similar to this rule. In [14] DWDL was compared with 2PL and the wound-wait CC method [21]. It was shown that DWDL outperforms these two CC methods to a significant degree.

A chief weakness of DWDL is its high communication overhead. Since the objective of DWDL is to limit the wait-depth of blocked transactions to some value (one) exactly, it has to operate on every wait-chain in the system directly. Since the lock conflicts that cause a wait-chain may occur at more than one node, it is impossible for DWDL to work without some centralized control nodes for maintaining the wait-chains. For DWDL, each time a lock conflict occurs at a local node, it is necessary to inform the centralized control nodes of this lock conflict event. Therefore, the number of messages increases rapidly as the level of data contention increases, which could impose a significant performance penalty.

3.3. The proposed LW DC\(_k\) method

In order to overcome the drawbacks of both 2PL and DWDL, LW DC\(_k\) is proposed. The idea of LW DC\(_k\) is as follows. Assume that a transaction encounters a lock conflict
with another transaction at a PE (recall that a transaction is implemented by sibling threads executed at PEs, and note that sibling threads can know the age (see Section 3.2 for the definition of transaction age) of the corresponding transaction). If the lock requester is at least \( g \) (\( g > 0 \)) years younger than the lock holder, then the lock requester waits for the lock holder; otherwise, one of them is restarted. Since like humans, in a given TP environment transactions have a fixed maximal life-span (say, \( \text{age}_{\text{max}} \)), note that the life-span of a transaction is its age when it is committed or aborted, and note that \( \text{age}_{\text{max}} \) decreases to some extent with increasing \( g \), the length of wait-chains cannot be unlimited for a TP system with LW DC\(_k\). Especially, we can adjust \( g \) to control the length of wait-chains. Therefore, unlike DWDL, LW DC\(_k\) can control the wait-depth of blocked transactions without the centralized control nodes. Next we present LW DC\(_k\) formally. Note that for LW DC\(_k\), \( g \) is not a constant so that the length of wait-chains can be controlled more efficiently. Also note that like humans, the transaction density decreases as the variable \( \text{age} \) approaches \( \text{age}_{\text{max}} \).

Assume that transaction \( T_r \) encounters a lock conflict with transaction \( T_h \), LW DC\(_k\) is a locking-based CC method with the following lock conflict resolution strategy:

\[
\text{if } L(T_h) > \alpha \ast L(T_r) + \beta, \text{ then } T_r \text{ waits for } T_h, \\
\text{else abort one of } T_r \text{ and } T_h.
\]

Where \( \alpha \) and \( \beta \) are two coefficients, by which the maximal allowable wait-depth of blocked transactions can be controlled. In order to guarantee that \( T_r \) waits for only such \( T_h \) that is older than \( T_r \), it is required that \( \alpha > 1.0 \) and \( \beta \geq 0.0 \). Assume that the ultimate aim is to limit the wait-depth of blocked transactions to no more than one, we can derive the values of \( \alpha \) and \( \beta \) as follows. Note that by the above definition of LW DC\(_k\), it is obvious that LW DC\(_k\) is deadlock-free.

Suppose that there exists a wait-chain with a length of two, say \( T_2 \to T_1 \to T_0 \). There should have:

\[
L(T_0) > \alpha \ast L(T_1) + \beta \\
L(T_1) > \alpha \ast L(T_2) + \beta
\]

This follows that

\[
L(T_0) > \alpha \ast \left[ \alpha \ast L(T_2) - \beta \right] + \beta \equiv L(T_0) > \alpha^2 \ast L(T_2) + \beta(1 + \alpha)
\]

Considering that \( \alpha \) must be larger than 1.0, a small \( \alpha \) helps to reduce restart rate, and small restart rate benefits system performance, we assign \( \alpha \) a relatively small value of 1.05. This value may not be optimal, but, is enough to make LW DC\(_k\) function well by our simulation experiments. Then there should have

\[
L(T_0) > 1.1025L(T_2) + 2.05\beta
\]

Let the right side of the above inequality be equal to a very large value, denoted by \( L_{\text{rare}} \), then such a \( \beta \) can be obtained that ensures that few transactions can become candidates for \( T_0 \) potentially for any given \( T_2 \). Considering that \( T_2 \) may encounter a lock conflict with \( T_1 \) at any time, let \( L(T_2) \) be zero, then \( \beta \) is fixed at \( L_{\text{rare}}/2.05 \), which ensures that the wait-depth of blocked transactions can be limited to no more than one approximately. Assume that the statistics of transaction running time can be obtained
that is defined as follows. Given a transaction \( T \), if \( T \) is committed (resp. aborted), then its running time is the difference of the committing (resp. aborting) time and its starting time (for its latest invocation if \( T \) is restarted). Then the wait-depth of blocked transactions can be limited to no more than one with a given probability. For example, let \( L_{\text{rare}} \) be the 95th percentile of the transaction running time, denoted by \( L_{95} \). Then the occurrence probability of such a transaction whose age is not less than \( L_{95} \) is 0.05, and thus the occurrence probability of wait-chains longer than one is 0.05, that longer than two is expected to be less than 0.05.

What we have stated above is illustrated in Fig. 6(a), where the horizontal axis corresponds to the age of a lock requester \( T_{r} \), and the vertical axis corresponds to the age of another transaction \( T_{h} \) with which \( T_{r} \) encounters a lock conflict.

Now let us consider the problem ‘Abort one of \( T_{r} \) and \( T_{h} \)’ in Fig. 6(a). Furthermore we distinguish two basic classes of transactions: read-only (RO) and read-write (RW) transactions. An RO (resp. RW) transaction is a collection of read (resp. update) actions on the database. Let \( T_{1}^{rw} \) and \( T_{2}^{rw} \) (resp. \( T_{1}^{ro} \) and \( T_{2}^{ro} \)) be two RW (resp. RO) transactions. Between these two classes of transactions there exist three conflict patterns as follows. Corresponding to these conflict patterns we have the following victim selecting rules, where the symbol ‘\( \rightarrow \)’ is used to represent the relation ‘conflict with’:

1. \( T_{1}^{rw} \rightarrow T_{2}^{rw} \) if \( L(T_{2}^{rw}) > L(T_{1}^{rw}) \) then abort \( (T_{1}^{rw}) \) else abort \( (T_{2}^{rw}) \),
2. \( T_{1}^{rw} \rightarrow T_{2}^{ro} \) if \( L(T_{2}^{ro}) > k * L(T_{1}^{rw}) \) then abort \( (T_{1}^{rw}) \) else abort \( (T_{2}^{ro}) \),
3. \( T_{1}^{ro} \rightarrow T_{2}^{rw} \) if \( L(T_{2}^{rw}) > 1/k * L(T_{1}^{ro}) \) then abort \( (T_{1}^{ro}) \) else abort \( (T_{2}^{rw}) \).

If two RW transactions conflict with one another, then the younger one is aborted so as to minimize the wasted work; otherwise the factor \( k \) is used to adjust the ratio of the abort rates of the RW and the RO transactions. \( k \) is expressed as \( \rho(RT_{rw} * RR_{rw} * ML_{rw})/(RT_{ro} * RR_{ro} * ML_{ro}) \), where \( RT_{rw} \) (resp. \( RT_{ro} \)) is the average response time of RW (resp. RO) transactions; \( RR_{rw} \) (resp. \( RR_{ro} \)) is the average restart ratio defined as the average restart number per RW (resp. RO) transaction; \( ML_{rw} \) (resp. \( ML_{ro} \)) is the average number of granules required by each RW (resp. RO) transaction. \( \rho \) is a factor used to incline the abort to the RW transactions since the RW transactions are more likely to block other transactions than the RO ones do. We found a good choice for \( \rho \) is about 0.45.
In the following we discuss qualitatively the features of LW DC_k by discussing its parameter L_{rare}, giving some cases where LW DC_k would not work well, and presenting a comparison of LW DC_k and DWDL. The whole algorithm is depicted in Fig. 6(b). As a system-related parameter, in this paper L_{rare} is held constant. In determining L_{rare} the tradeoff between the overhead of restarts and the benefit from restarts has to be taken into account. A large L_{rare} incurs a high restart rate which has negative effect on system performance, but leads to short wait-chains which benefits system performance. A large L_{rare}, therefore, is beneficial to a TP system with a high level of data contention. In fact, it is better for L_{rare} to be adjusted dynamically according to the current data contention level. This will eventually adjust the maximal allowable wait-depth of blocked transactions. Some more discussion about L_{rare} will be given in Section 4.2.3.

As all the other restart-oriented CC methods, LW DC_k has some inevitable drawbacks. On the one hand, in LW DC_k the transactions are required not to interact with users by themselves; otherwise users would be involved in the restart processing of their transactions and become tired of using the system. On the other hand, if the level of data contention is not high enough, restarting transactions is unnecessary and may be harmful to system performance. Therefore, it may be better to use 2PL in the normal case, and to use LW DC_k only when the level of data contention becomes high enough.

Now let us compare LW DC_k and DWDL simply. As described above, both LW DC_k and DWDL limit the wait-depth of blocked transactions. However, LW DC_k is a distributed approach that uses the local information of a node, whereas DWDL is a centralized approach that uses the overall information of the network. One of the advantages of the distributed approach is that the communication overhead can be reduced largely. In the distributed approach, however, the wait-chains can no longer be operated directly, and thus the wait depth of blocked transactions cannot be controlled exactly and efficiently.

4. A simulation experiment

In this section we present a simulation experiment which is based on the simulation system introduced in Section 2 and the CC methods introduced in Section 3. For the current version of our simulation system, the costs of the CPU requests are given by time requirements although we think it may be convenient for them to be calculated by using processor speed and the average number of instructions per request.

4.1. Experiment environment

For the experiments here it is assumed that each relation of database is horizontally partitioned among all the PEs uniformly, all records of a relation are equally likely accessed, and the granules accessed by a transaction are distributed uniformly over all the disks at a PE. The parameter settings are shown in Tables 1–4. Four kinds of transactions are included in our workload: update, select, join-2, and join-3, which are structured as relational algebra query trees (see Section 2.2). The update contributes to the RW transactions. The select, join-2, and join-3 contribute to the RO transactions.
Table 1
Workload related parameters

<table>
<thead>
<tr>
<th>Transaction type: update, select, join-2, join-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relation number accessed per transaction: 1, 1, 2, 3</td>
</tr>
<tr>
<td>Partition number accessed per relation (G): 3, 16, 16, 16</td>
</tr>
<tr>
<td>Granule number accessed per partition (G): 2, 6, 6, 6</td>
</tr>
<tr>
<td>Occurrence frequencies of transactions:</td>
</tr>
<tr>
<td>(i) 0.3, 0.5, 0.15, 0.05</td>
</tr>
<tr>
<td>(ii) 0.7, 0.214, 0.064, 0.022</td>
</tr>
</tbody>
</table>

G: geometric.

The select (resp. update) is a collection of read (resp. update) accesses to a relation; the join-2 (resp. join-3) reads records from two (resp. three) relations and performs join operation over them. For join-2 and join-3 we distinguish I/O components for reading records and CPU components for joining these records. All these components are executed concurrently. CPU components have data dependencies on I/O components. The updated data are assumed to reside in the same PE as they did. Assume that the memory is battery backed up, and thus no physical disk logging is performed for updates. Two kinds of combination of occurrence frequencies of RW and RO transactions (0.3 vs. 0.7 and 0.7 vs. 0.3) are used so as to distinguish the query-bound and the update-bound application environments.

Global deadlock detection interval is varied from 100 to 500 ms adaptively. Each time the deadlock detection process is started, if a deadlock is found, this interval is increased by 5 ms; otherwise decreased by 5 ms. The input data to a component is assumed to be processed and sent to its successors without loss. If a transaction is restarted, the costs for setting up a transaction and its sibling threads are halves of what are given in Table 4. The $L_{rare}$ is set to be 10th percentile of the transaction running time. The costs for commit in Table 4 include those for pre-committing and completing transactions.

The proposed method is compared with 2PL and DWDL methods. The simulation experiments are performed under the assumption that sufficient front-end nodes have been configured. That is, the processing requests at front-end nodes, such as network communication, restart, commit, and transaction initiation, can be started without delay. This in effect gives favored treatment to DWDL, since it shows the highest restart rate. The simulation experiments were also performed by limiting the number of front-end

Table 2
Database related parameters

<table>
<thead>
<tr>
<th>Number of relations in the database: 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of records per relation: 32000</td>
</tr>
<tr>
<td>Partition number per relation: 32</td>
</tr>
<tr>
<td>Number of records per partition: 1000</td>
</tr>
<tr>
<td>Record size in byte: 200</td>
</tr>
<tr>
<td>Lock granularity (record number per lock): 1</td>
</tr>
</tbody>
</table>
Table 3
Computer system related parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of processing elements in the system</td>
<td>32</td>
</tr>
<tr>
<td>Number of disks per processing element</td>
<td>4</td>
</tr>
<tr>
<td>Network speed</td>
<td>2.22 MByte/s</td>
</tr>
<tr>
<td>CPU cost for sending/receiving a message</td>
<td>0.05 ms</td>
</tr>
<tr>
<td>Control/data message size</td>
<td>512/4096 bytes</td>
</tr>
<tr>
<td>I/O time for accessing a granule from disk</td>
<td>20.0 ms</td>
</tr>
<tr>
<td>Hit ratio of database cache</td>
<td>60%</td>
</tr>
</tbody>
</table>

nodes, by setting the number of disks per PE to ten, or by ignoring the database cache. The results, however, are not reported here for the sake of brevity. In all these experiments, LW DC_k outperforms both 2PL and DWDL.

Two primary performance metrics with which we are concerned are the system throughput that is defined to be the transaction completion rate, and the restart ratio that is defined to be the average number of transaction restarts per commit. The average transaction response time can be obtained easily by Little's law.

The simulation results are given in Section 4.2. For each simulation result, the 95% confidence intervals of measured indices are within ±5% of the mean of the indices.

4.2. Results and discussion

In this subsection we present and discuss the simulation results for 2PL, DWDL and the proposed method.

4.2.1. System throughput

System throughput characteristics are shown in Fig. 7, where two application environments are included: the query-bound (identified by Q) and the update-bound (identified by U) environment.

When MPL is very low, all the CC methods show almost the same throughput characteristics, since lock conflicts seldom occur, and CC methods have little effect on throughput. As MPL increases, the probability of lock conflicts is also increased. For 2PL, the transactions that encounter lock conflicts are always blocked unless deadlocks

Table 4
Transaction processing related parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deadlock detection interval (for 2PL)</td>
<td>100–500 ms</td>
</tr>
<tr>
<td>Mean restart delay (E)</td>
<td>500 ms</td>
</tr>
<tr>
<td>CPU cost per granule for I/O component</td>
<td>0.05 ms</td>
</tr>
<tr>
<td>CPU cost per granule for CPU component</td>
<td>0.01 ms</td>
</tr>
<tr>
<td>CPU cost for setting up a transaction/sibling</td>
<td>1.0/0.1 ms</td>
</tr>
<tr>
<td>CPU cost for commit at TM/PE</td>
<td>0.35/0.26 ms</td>
</tr>
<tr>
<td>CPU cost for abort at TM/PE</td>
<td>0.12/0.10 ms</td>
</tr>
</tbody>
</table>

E: Exponential.
occur. If the level of the data contention is not high enough, blocking a transaction that encounters a lock conflict is preferable to restarting a transaction that involves a lock conflict. As MPL increases further, however, data contention becomes high enough to degrade system performance. At this time, it is better to adequately restart some transactions which involve lock conflicts.

DWDL improves system performance by limiting the wait-depth of blocked transactions to one by selecting some transactions to restart. The aim is to increase the number of the active transactions as far as possible. As shown in Fig. 7, DWDL outperforms 2PL greatly.

Because the proposed method also limits the wait-depth of blocked transactions, it outperforms 2PL too. Its performance benefit, however, is obtained at a far lower restart cost than that of DWDL, as shown in Section 4.2.2. We argue that a low restart ratio is preferable. Restarting a transaction causes a lot of work to be wasted. The wasted work includes at least: (1) initiating a transaction and its sibling threads; (2) sending intermediate results and other kinds of messages; and (3) processing data, involving lock requests, granule accesses to disks, and granule processing by CPU. In TP systems, the fact that disk I/O is the main bottleneck has been recognized by numerous studies in the past. It is more than likely that in the foreseeable future, this situation will not change. Therefore a high restart ratio is still unacceptable although the processor speed has been improved greatly. In addition, we found that releasing the limitation on the wait-depth of blocked transactions properly may be better than limiting it to one. For the experiments here, the $L_{\text{rate}}$ is set to be only 10th percentile of the transaction running time. As shown in Fig. 7, the proposed method outperforms DWDL in both the maximal possible throughput and the thrashing point (i.e., the MPL at which thrashing occurs).

We observed that DWDL is apt to abort the RW transactions in our experiments. Therefore, for DWDL the average response time of the RW transactions is not proportional to their short average size. In fact, for DWDL the average response time of the RW transactions increases rapidly as MPL increases, and would be greater than that of the RO transactions, although the number of granules required by each RW
transaction is far less than that required by each RO transaction. For LW DC, RW and RO transactions are restarted in more appropriate ratio, and a short response time is guaranteed for both of them. That is why the performance difference of DWDL and LW DC for query-bound environment is different from that for update-bound environment in Fig. 7.

4.2.2. Restart ratio

Restart ratios are illustrated in Fig. 8. In this Fig. 2PL shows the ideal restart characteristics. This is because the deadlock, which is the only cause of restarts for 2PL, is rare by numerous 2PL performance studies (e.g., [28]). System performance, however, is not determined by its restart ratio only. What makes a TP system behave well is a tradeoff between blocking and restarts. Restart ratio is higher for DWDL and the proposed method than for 2PL, throughput characteristics of them, however, are also superior to that of 2PL. This becomes even more obvious as data contention becomes serious. Naturally, if data contention is very weak, excessive restarts degrade system performance since a lot of work is wasted.

For the proposed method, we found that when MPL is very low, its restart ratio is slightly higher than that of DWDL. The reason is that in this case, wait-chains longer than one seldom occur, the abort criterion of the proposed method, however, is met occasionally. Note that the \( L_{\text{rare}} \) is now set to be 10th percentile of the transaction running time. In Section 4.2.3 we will show that in the case of low MPL (i.e., low level of data contention), a small \( L_{\text{rare}} \) should be employed, which helps to reduce restart ratio.

As MPL becomes slightly higher, however, wait-chains longer than one occur frequently since the parallel processing tends to cause a high lock conflict rate. At that time, restart ratio with DWDL grows rapidly. Restart ratio with the proposed method, however, increases in a moderate way. The reason is that for the proposed method, the maximal wait-depth of blocked transactions is allowed to be larger than one, and the characteristics of the transactions and the properties of the parallel transaction processing are taken into consideration.
4.2.3. Restart ratio, throughput, and $L_{\text{rare}}$

Figs. 9 and 10 show the effect of $L_{\text{rare}}$ on restart ratio and throughput respectively in the update bound application environment. Note that for the query-bound application environment, similar results can also be obtained. Here we use the MPL of 90 and 250 to represent the application environments with moderate and serious data contention, respectively.

Fig. 9 shows that restart ratio increases as $L_{\text{rare}}$ increases. This coincides with what was shown in Fig. 6(a), where the area marked by ‘Abort one of $T_i$ and $T_h$’ becomes large as $L_{\text{rare}}$ increases.

As shown in Fig. 10, for a system with a given level of data contention, throughput increases as $L_{\text{rare}}$ rises up to some value, and then it decreases. There is an optimal value of $L_{\text{rare}}$ that gives the maximal throughput. The reason is as follows. Here throughput is affected mainly by the average length of wait-chains and the restart rate. As $L_{\text{rare}}$ increases, the average length of wait-chains decreases, which benefits throughput;
whereas the restart rate increases, which has a negative effect on throughput. If $L_{\text{rare}}$ is not large enough, the restart rate is small, but the average length of wait-chains is long, and thus the data contention dominates throughput. At this time, increasing $L_{\text{rare}}$ helps to increase throughput. When $L_{\text{rare}}$ becomes large enough, and thus the restart rate becomes dominant, throughput decreases with increasing $L_{\text{rare}}$. From the above discussion we also see that a high level of data contention corresponds to a large optimal value of $L_{\text{rare}}$, what is shown in Fig. 10 supports this result.

According to the above discussion, we suggest that for a given system, the value of $L_{\text{rare}}$ be determined according to the level of data contention. A table for mapping the level of data contention onto the value of $L_{\text{rare}}$ can be obtained by simulation, by experiments on this system, or by experience of the system manager in using this system.

5. Conclusion

We have presented a simulation system for the shared-nothing parallel TP systems, which aims at, but is not restricted to, the studies of the concurrency control methods. In this simulation system, transactions are represented by their dataflow graphs. It was argued that such a workload model can be used for modeling most application environments, and is specially suitable for the studies of the concurrency control methods for the parallel TP systems. Many mature simulation techniques have been integrated organically in this simulation system, by which dataflow-oriented shared-nothing parallel TP systems were simulated correctly and efficiently. In addition to the concurrency control methods, many other performance factors can also be studied by using this simulation system, some examples of which are the degree of system parallelism, the level of data and resource contention, the CPU/Disks scheduling strategy, the deadlock detection/resolution method, and the transaction restart policy. As an application example, this simulation system was used to evaluate three concurrency control methods, which are 2PL, DWDL, and the proposed LW DC$_k$. Initial use of this simulation system demonstrates its effectiveness.

We have also proposed a restart-oriented locking-based concurrency control method called LW DC$_k$ for shared-nothing parallel TP systems. LW DC$_k$ has such a property that it ensures the wait-depth of blocked transactions is limited. Numerous performance studies concerning 2PL have shown that unlimited waiting degrades system performance. Therefore, TP systems with LW DC$_k$ would behave well. A unique characteristic of LW DC$_k$ is that it limits the wait-depth of blocked transactions locally. In addition, LW DC$_k$ can prevent the occurrence of deadlocks. Compared to the 2PL, LW DC$_k$ ensures that TP systems can work well in a wide spectrum of data contention due to its lock conflict resolution mechanism. Compared to the DWDL, on the one hand, LW DC$_k$ shows far lower communication overhead since the restart decisions are made locally; on the other hand, LW DC$_k$ shows far lower restart ratio (unless the data contention is very weak), since the characteristics of the transactions and the properties of the parallel transaction processing are taken into consideration. Simulation results showed that LW DC$_k$ outperforms 2PL and DWDL.
Now we can generate dataflow graphs with arbitrary topologies. Some dataflow graphs, however, may correspond to unreal transactions. Future work includes investigating the relationship between real-life transactions and dataflow graphs by examining all the existing database benchmarks. The objective is a demand-driven workload generator. Also planned as the future work is to find an adaptive strategy to adjust $L_{\text{rare}}$ dynamically according to the current data contention level, and perform a variety of simulation experiments to investigate the performance behavior of LW DC$_k$ more deeply.

References