Buffer Reduction in an Attribute-Based
Concurrent Transaction
Processing System

Lawrence. J. Henschen
Northwestern University
Dept. of EECS 2145 Sheridan Rd.
Evanston IL 60208-3103
henschen@eecs.nwu.edu

Julia C. Lee*
Argonne National Laboratory
9700 S. Cass Ave. DIS/900
Argonne IL 60493-4832
lee@dis.anl.gov

ABSTRACT
We presented a way to improve the buffer usage for the transaction management model in an earlier study. The method presented in this paper identifies the conflicting and the non-conflicting parts of the affected data sets and separates them. The original operations are converted into two sets of operations - one set operates on disjoint data and can be executed in parallel, another set operates on conflicting data using buffers. This approach will reduce the size of the buffers used in the earlier approach.

Keywords: database, transactions, concurrency, conflicts, specification, unifiable

1. Introduction

In [HL94] [HL95] we presented a new transaction management model by analyzing the semantics of the database operations. We studied the existing semantic approaches [Gar83][Lyn83][FO89] and proposed a new approach to analyze the semantics of the database operations. This new model allows the possibility of increasing the degree of concurrency substantially. Since the resulting new operations produced by the algorithms in our model will be totally de-synchronized, they can be executed in parallel. We think that this model presents a new approach for managing database transactions. Since the model uses buffers to resolve local and global conflicts, the buffers are important elements of the process. In our model there is a buffer for inserted tuples and one buffer to hold the tuples to

* The submitted manuscript has been authored by a contractor of the U.S. Government under contract No. W-31-109-ENG-38. Accordingly, the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes.
be changed by each UPDATE operation. These buffers can be de-allocated once the operation is done. This is an improvement to the traditional “multiversion” models [BHG87][KS88][AK91] since in the “Multiversion” models “Whenever a transaction attempts to write a data item, the system creates a new version of the data item with the new value” [KS88]. Also there needs to be a “version control” function in the traditional multiversion model to keep track of the different versions belonging to different operations and different transactions. However, in the model of [HL95] the buffers hold all of the affected and inserted tuples of all the INSERT and UPDATE operations of the transaction. In this paper we show how to improve this so that buffers hold only those tuples for which there is potential conflict among operations in the tuple. Those portions of the tuple sets referenced by a transaction for which there is no conflict at all be handled separately and in parallel. The set of tuples for which there is potential conflict can then be handled in the buffers according to the techniques of [HL95], and the buffer will be generally smaller. Moreover, this “factoring out” process can be applied to transaction processing to reduce the set of conflicting tuple sets even if the techniques of [HL95] are not used.

The rest of this article is arranged as follows: In Section 2 we introduce some of the definitions used by the following sections. These definitions may also be found in [HL94] [HL95]. Section 3 presents a more detailed study on the attribute sets and the data sets of the database operations. Section 4 presents the technique to factor out the nonconflicting parts of the data sets. Section 5 is the closing remark.

2. Definitions

**Definition 2.1:** A database state \((DBT)\) [Ull88] [MH89] consists of the set of tuples in the relations of the database.

In this paper we consider four database operation types: READ, INSERT, DELETE and UPDATE. There are also five corresponding buffer operation types: READB,
APPENDB, INSERTB, DELETEB and UPDATEB.

**Definition 2.2:** An attribute specification for a relation R of arity n is a sequence $(S_1, S_2, ..., S_n)$ where: each $S_i$ is either a constant or a variable. The set of constants in the sequence is denoted by $C$ and the set of variables by $V$. We will write $(C, V)$ when the particular order of the constants and variables is not crucial to the context. When C is empty (i.e., $(S_1, S_2, ..., S_n)$ consists of all variables) we say the specification is empty and abbreviate it as $()$. A specification with exceptions is an expression of the form $S - T$ where $S$ is a specification and $T$ is a set of tuples. In general, if $S_1$ and $S_2$ are specifications, we may also write $S_1 \cup S_2$ and $S_1 \cap S_2$ as complex specifications. For simplicity we will assume throughout this paper that the variables in a specification are all distinct; i.e., there are no repeated variables.

**Definition 2.3:** The set of tuples in R matched by an attribute specification $(S_1, S_2, ..., S_n) = (C, V)$ is the set of tuples in R which have the same values in the same attribute positions as the constants in $C$.

We denote the matched tuple set for a specification $S$ in relation $R$ by $MS_S(R)$. Obviously, the matched tuple set depends on the database state. The set of tuples in R matched by $S - T$ is the set difference $MS_S(R) - T$. The set of tuples matched by $S_1 \cup S_2$ ($S_1 \cap S_2$) is the set union (intersection) of those matched by $S_1$ and $S_2$. The set of the tuples matched by $()$ - empty specification - is the entire tuple set for the relation. We use $S_{\emptyset}$ to denote the specification which matches the empty tuple set.

**Definition 2.4:** A Database operation is a four-tuple $O = (OP, TDBT, A, B)$ where

1. $OP$ is one of the ordinary database operations we are concerned with (READ, INSERT, DELETE, UPDATE);
2. $TDBT$ is the target database state on which this database operation is applied. Since we assume that any operation affects only one relation at a time, $TDBT$ is a relation name in general. As we will see in the next definition, $TDBT$ could also be a buffer.

**DISCLAIMER**

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.
DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.
3. \( A \) is a pair of attribute specifications for the affected relation written as \(((C_{in}, V_{in}), (C_{out}, V_{out}))\) or simply \((S_{in}, S_{out})\).

4. \( B \) is a pair \(((C_B, V_B), BOL)\) or \((S_B, BOL)\) where \(S_B\) is the specification of the buffer content which should have the same form as in Definition 2.2, and \(BOL\) is a list of the buffer operations associated with this buffer (or this operation). \(BOL\) could contain 0 or more of buffer operations mentioned above. The formal definition of the buffer operations will be given in Definition 2.5.

5. There will always be 0 or one READB followed by 0 or more APPENDB operations at the beginning of the list of the buffer operations if \(BOL \neq \emptyset\). READB and APPENDB are the operations to load the buffer. They will be executed prior to any other buffer operations and prior to the execution of the all the ordinary operations. That is, buffer-loading is done at the beginning of the transaction prior to any actual database operation taking place. The rest of the buffer operations can be executed any time prior to the execution of the operation which owns the buffer.

Note: in general the database operations submitted by an original transaction do not have buffer operations. That is \(BOL = \emptyset\) for the operations submitted by a transaction. The transaction management model we introduced in [HL95] will create new database operations with actual buffer operations. For uniformity we use the same definition for both original database operation and the newly created database operation.

**Definition 2.5:** A buffer operation is a three-tuple \( O = (OP, TDBT, A) \) where

1. \( OP \) is one of the buffer operations we are concerned with (READB, APPENDB, INSERTB, DELETEB, UPDATEB);
2. \( TDBT \) is the name of the buffer.
3. \( A \) is the same as in Definition 2.4;

Note that we assume that a buffer operation does not, in turn, contain any associated buffer operations.
For example: We could have an operation submitted by a transaction T of the form (UPDATE, R, (S₁, S₂), (S₂, Ø)) in which case there are no buffer operations. We could also have a newly created database operation like:

(INSERT, R, (S₀, S₃), (S₃, BOL)) where BOL is:

[(READB, Bᵢ, (S₄, S₄))

(DELETEB, Bᵢ, (S₅, S₀))

(UPDATEB, Bᵢ, (S₄, S₃))]

in which case the (INSERT, R, ...) has an associated list of buffer operations BOL. In this case tuples matching S₄ are loaded into Bᵢ at the beginning of the transaction containing this INSERT operation. Then DELETEB and UPDATEB are performed in order, after which tuples remaining in Bᵢ that match S₃ are inserted into R.

Note that we should have Sᵢᵣ = Sᵢᵩ for READ and READB, Sᵢᵣ = S₀ for INSERT and INSERTB, Sᵢᵩ = S₀ for DELETE and DELETEB.

Definition 2.6: The input database state of a database operation is the database state before the database operation is performed

Definition 2.7: The affected set AS₀ of a database operation O is the set of tuples in R matched by Sᵢᵣ just before O is performed.

Note that AS₀ = MSᵢᵣ( DBT ) .

Definition 2.8: The resulting set RS₀ of a database operation O is the set of tuples inserted into the database by O during the execution of O, and it matches Sᵢᵩ. Note that for the case of UPDATE or UPDATEB, RS₀ = MPₜᵢᵩ( AS₀ ) . This notation states that the result set is a mapped set from the affected set AS₀ . The mapping (or set operation) is defined by Sᵢᵣ and Sᵢᵩ of O.

Definition 2.9: Specification S₁ subsumes specification S₂ if for every position in S₁ containing a constant the corresponding position in S₂ also contains the same constant.

For example (0727, x, y, z, Henschen) subsumes (0727, C49, u, v, Henschen). When
tuple specification $S_1$ subsumes tuple specification $S_2$, the tuple set matching $S_2$ is contained in the tuple set matching $S_1$.

**Definition 2.10:** The output state of a database operation is the database state after the operation is performed and is denoted $O(DBT)$ where $DBT$ is the input database state.

Notice that $O(DBT) = (DBT - AS_o) \cup RS_o$. Also, if $O$ is a retrieval operation (i.e., the OP is READ) $O(DBT) = DBT$. We will also use the notation $AS_o(DBT)$ to refer specifically to the affected set of an operation $O$ applied to a specific input database state $DBT$.

3. **Attribute sets and the data sets**

The method introduced in [HL95] and the extension to be developed here depends heavily on comparing the attribute specifications and corresponding matching tuple sets for two operations. We therefore describe the matching process in more detail. The concept is a straightforward application of unification in automated reasoning and its application to deductive databases, as in [HN84] and [MH89]. In this description we will not be interested in the particular order of the attributes in the specifications and will usually write the attributes which are constants at the beginning of the list. Of course, in practice, the constants in a specification can be in any position.

Consider two attribute specifications

$$S_1 = (a, b, x, c, y)$$

$$S_2 = (a, d, e, z, u)$$

where

1. $a$ represents the constant vectors common to both $S_1$ and $S_2$;
2. $b$ and $d$ are constant vectors having the same length and attribute position but different values for $S_1$ and $S_2$;
3. $e$ is a vector of constants, and $x$ is a vector of variables of the same length and
attribute positions

4. $z$ is a vector of variables, and $c$ is a vector of constants of the same length and attribute positions and

5. $y$ and $u$ are variable vectors of the same length and attribute positions.

For a more concrete example let us consider two attribute specifications

$(385, 5/20/93, x_3, x_4, x_5, x_6, \text{Chicago})$ and

$(385, 5/20/93, 1000, y_4, y_5, y_6, y_7)$.

In this case, $a$ is $(385, 5/20/93)$, $c$ is (Chicago), $e$ is (1000), $x$ is $(x_3)$, $y$ is $(x_4, x_5, x_6)$, $z$ is $(y_7)$, and $u$ is $(y_4, y_5, y_6)$. In this example, $b$ and $d$ have length 0.

Let us now suppose that the variables in $S_1$ are also distinct from those in $S_2$. Clearly, if $b$ and $d$ have length > 0, there can be no tuples in common in the matching tuple sets. For example, suppose $b$ specifies the attribute NAME to be HENSCHEN and $d$ specifies the attribute NAME to be LEE; obviously, no tuple can have both values for the NAME attribute. (In automated reasoning and deductive databases, $S_1$ and $S_2$ would be called non-unifiable in this case.) On the other hand, if $b$ and $d$ have length 0, as in the two specifications in the preceding paragraph, then the matching tuple sets may or may not have tuples in common depending on $a$, $c$, and $e$, the variables and the current database state. (In this case, $S_1$ and $S_2$ would be called unifiable in automated reasoning terms.) In such cases there may be conflict between the operations containing $S_1$ and $S_2$ depending on where $S_1$ and $S_2$ occur in the two operations (i.e., specifying AS or RS) and what the operations themselves are.

It is easy to derive a specification $S_3$ that describes the potential conflicts of two specifications $S_1$ and $S_2$ as the following theorem shows.

**Theorem 3.1:** The specification of the conflict set of two matched sets can be derived from the specifications of the two set $S_1$ and $S_2$.

**Proof:**
\[ S_1 = (a, b, x, c, y) \text{ and } S_2 = (a, d, e, z, u) \] where \( a, b, x, c, y, d, e, z, u \) are as we defined above.

Let us denote the length (number of attributes) of a vector \( v \) as \(|v|\).

**Case 1:** if \(|b| = |d| > 0\)

Then it is obvious that \( MS_{S_1} (DBT) \cap MS_{S_2} (DBT) = \emptyset \).

**Case 2:** if \(|b| = |d| = 0\). In general we have \( S_1 = (a, x, c, y) \) and \( S_2 = (a, e, z, u) \).

Clearly,

\[ MS_{S_1} (DBT) \cap MS_{S_2} (DBT) = MS_{S_3} (DBT) \] where
\[ S_3 = (a, e, c, v) \]

There are many special cases when one or more of the relevant vectors has length 0.

For example, if \( |c| = 0 \), then \( S_3 = (a, e, v) \); if \( |e| = 0 \), then \( S_3 = (a, c, v) \), etc.

Q. E. D.

Note that in Case 2, \( S_3 \) is just the common instance of \( S_1 \) and \( S_2 \) after unification ([HN84], [MH89]).

Note that the vectors of constants and variables \( a, b, c, d, e, x, y, z \) and \( u \) may be inter-spersed in \( S_1 \) and \( S_2 \). However, a simple one-pass, left-to-right scan is sufficient to identify them and form \( S_3 \). The following linear algorithm suffices and is essentially the normal unification algorithm of automated reasoning adapted to the special case of only variables and constants. Let \( k \) be the number of attributes in the specifications. Let \( S(i) \) refer to the \( i \)th attribute of the specification \( S \).

\[
\text{for } (i=1; i<=k; i=i+1) \{
 \text{if ( } S_1 (i) \text{ and } S_2 (i) \text{ are different constants )} \\
 \{ 
 \quad S_3 = S_\emptyset; \\
 \quad \text{break;} \\
 \} \\
\text{if ( } S_j (i) \text{ is a constant for } j=1 \text{ or } 2 \text{ )}
\]

\[ S_3(i) = \text{that constant}; \]
else
\[ S_3(i) = \text{a distinct new variable}; \]
\]

In case the variables in a specification need not be all distinct, a modified algorithm which records connections between variables and constants discovered in the loop will suffice and is also still linear.

4. Conflicting and non-conflicting parts of transactions

Suppose we have a sequence of operations whose selecting specifications are \( S_1, S_2, \ldots, S_n \). If we pairwise find the common instances as in Theorem 3.1, each \( S_i \) could be transformed into two parts - an \( S'_i \) with exceptions, the exceptions being anything that might lead to common tuple with some other \( S_j \), and an \( S''_i \) which is the part of \( S_i \) that unifies with at least one other \( S_j \). We will illustrate this idea in the next example. At that point we could parallelize the parts specified by the \( S'_i \) because none of these would have conflict with ANY OTHER specification. That would leave just the sequence

\[
O_1 S''_1 \\
O_2 S''_2 \\
\vdots \\
O_n S''_n
\]

for which conflict needs to be handled either by the buffer approach of [HL94] and [HL95] or by any of the other traditional methods.

Here’s an example.

\( O_1 \) : (INSERT, R, \( (S_0, (e, f, g)) \), B1)

\( O_2 \) : (INSERT, R, \( (S_0, (a, b, c)) \), B2)

\( O_3 \) : (DELETE, R, \( ((a, x, c), S_0) \), B3)
$O_4$: (UPDATE, R, ((y, b, c), (y, h, c)), B4)  

$O_5$: (DELETE, R, ((u, d, c), S₀), B5)

$O_1$ does not unify with anything else and can be done totally independently from all the others and in parallel. $O_2$ potentially conflicts with $O_3$ and $O_4$, but not with $O_5$. (In this case, actually $O_2$ should just be removed because $(a, b, c)$ will definitely be deleted, but let us just go on with the example.) However, the only point of conflict is the tuple $(a, b, c)$. For $O_3$ this means any tuple matching $(a, x, c)$ but with $x$ different than $b$ will not be in conflict with $O_2$, and for $O_4$ any tuple matching $(y, b, c)$ but with $y$ different than $a$ will also not be in conflict with $O_2$. Note that $O_2$ and $O_5$ are not in conflict at all. Now compare $O_3$ with $O_4$. $O_3$ and $O_4$ could conflict for $y = a$ and either $x = b$ or $x = h$. But any other combination will not be in conflict. Similarly, $O_3$ and $O_5$ could be in conflict for $u = a$ and $x = d$. Finally $O_4$ and $O_5$ are not in conflict at all because of the mismatch of the second attribute for both $(y, b, c)$ and $(y, h, c)$. That means that for the sets of tuples in the operations:

$O_1$: (INSERT, R, (S₀, (e, f, g)), B1) with no exceptions  

$O_2$: (INSERT, R, (S₀, (a, b, c)), B2) with the exception $(a, b, c)$,  
i.e., the tuple $(a, b, c)$ has to be handled through the buffer or other means of handling conflict.  

$O_3$: (DELETE, R, ((a, x, c), S₀), B3) except for $x = b$ or $x = h$ or $x = d$  

$O_4$: (UPDATE, R, ((y, b, c), (y, h, c)), B4) except $y = a$  

$O_5$: (DELETE, R, ((u, d, c), S₀), B5) except $u = a$

Then we could do in parallel:

$\sigma''_1$      $\sigma'_1$      $\sigma'_2$      $\sigma'_3$      $\sigma'_4$      $\sigma'_5$

$\sigma''_2$

$\sigma''_3$

$\sigma''_4$
where $\sigma_1$ is (INSERT, R, $(S_\emptyset, (e, f, g))$, B1)

$\sigma_2$ is NULL operation since

$((\text{INSERT}, R, (S_\emptyset, (a, b, c)), B2)$ with the exception $(a, b, c))

$\sigma_3$ is (DELETE, R, $((a, x, c), S)$, B3) except for $x$ in the set \{b, h, d\}

$\sigma_4$ is (UPDATE, R, $((y, b, c), (y, h, c))$, B4) except for $y$ in the set \{a\}

$\sigma_5$ is (DELETE, R, $((u, d, c), S_\emptyset)$, B5) except for $u$ in the set \{a\}

and $\sigma''_2$ is NULL

$\sigma''_3$ is (DELETEB, B, $((a, b, c), S_\emptyset)$)

$\sigma''_4$ is (UPDATEB, B, $((a, b, c), (a, h, c))$, B4)

$\sigma''_5$ is (DELETEB, B, $((a, d, c), S_\emptyset)$)

In this particular case the result is extremely simple because all the potentially conflicting tuples are fully grounded, that is there are no variables left at all in the exceptions. This needn't always be the case. For example, we could have had in this example 4 attributes and the last one was always a variable vector - i.e.

$$(e, f, g, w1)$$

$$(a, b, c, w2)$$

$$(a, x, c, w3)$$

etc.

Still, it seems that the exceptions would be relatively small and easy to express because they come out of the unification of specifications, and these all have just variables and explicit constants. For this kind of unification, the unification process for one pair of specifications is linear. For a very small overhead of unifying all pairs of specifications in
a transaction we can factor out the parts that do not conflict.

For completeness we state the following theorem and corollary. The proofs are straightforward and are only sketched here.

**Theorem 4.1.** Given the notation introduced in this section, performing an operation $O$ on a database state $DBT$ is equivalent to performing $O'$ and $O^\ast$ on $DBT$ in arbitrary order.

Proof Sketch.

Let $(S_{in}, S_{out})$ be the specification for $O$ and $(S'_{in}, S'_{out})$ and $(S''_{in}, S''_{out})$ the specifications for $O'$ and $O^\ast$ respectively. Obviously the tuples matched by $S_{in}$ and $S'$, $S''_{in}$ are disjoint and together equal the set of tuples matched by $S_{in}$; Similarly for $S_{out}$. From this remark a simple analysis of the three possibilities for $O$ yields the result.

**Corollary 4.1.** For any transaction $T$, applying the technique described in this section and the technique of [HL95] produces a sequence of conflict-free operations which can be executed in parallel and whose execution always produces the same final database state as executing $T$ itself.

Proof Sketch.

By Theorem 4.1, each $O_i$ in $T$ is equivalent to $O_i$ plus $O_i'$. By construction, the tuple sets involved in the $O_i$ operations are all disjoint from each other and from the tuples involved in the $O_i'$ operations. These operations are therefore independent of the $O_i''$ operations and from each other and can safely be executed in parallel. The technique in [HL95] to be applied to the $O_i''$ operations is proven correct in [HL95], completing the proof of this corollary.

5. Remarks

We have introduced a new approach to handle the conflicting sets of affected data for the operations in a transaction. For the conflict resolution method of [HL94] and [HL95]
Further study can be made on how to handle the integrity constraints using a similar approach by analyzing the specifications of operations as introduced in this paper and [HL94], [HL95].

Work was supported by the U.S. Department of Energy under contract W-31-109-Eng-38.

References

Using Delayed Commitment in Locking Protocols for Real-time Databases;
Proceeding of the 1992 ACM SIGMOD;

[AK91] D. Agrawal, V. Krishnaswamy;
Using Multiversion Data for Non-interfering Execution of Write-Only Transactions;
Proceeding of the 1992 ACM SIGMOD;

[BHG87] P.A. Bernstein, V. Hadzilacos, N. Goodman;
Serializability Theory;
Concurrency Control and Recovery in Database Systems;

[BR92] B. R. Badrinath and Krithi Ramamritham;
Semantics-Based Concurrency Control Beyond Commutativity;
ACM Transactions on Databases;

[FO89] Abdel Aziz Farrag, M. Tamer Ozsu;
Using Semantic Knowledge of Transactions to Increase Concurrency;
ACM Transactions on Database Systems;
[Gar83] Hector Garcia-Molina;
Using Semantic Knowledge for Transaction Processing in a Distributed Database;
ACM transaction on Database Systems;

[HL94] Lawrence J. Henschen and Julia C. Lee
A Highly Concurrent Transaction Management Model
Proceeding of ICCI’94 International Conference on Computing & Information. Page 1426-1441; 1994

[HL95] Lawrence J. Henschen and Julia C. Lee
Resolving Conflict - A Semantic Approach to Transaction Management Model
Submitted to “Information Systems”
1995

[HN84] Lawrence J. Henschen and Shamin A. Naqvi
On Compiling Queries in Recursive First-Order Database;
Journal of the Association for Computing Machinery;

[KS88] Henry F. Korth, Gregory D. Speegle;
Formal Model of Correctness Without Serializability;
Proceeding of SIGMOD International Conference on Management of Data;

[LKS91] Eliezer Levy, Henry F. Korth, Abraham Sil
An Optimistic Commit Protocol for Distributed Transaction Management;
Proceeding of the 1992 ACM SIGMOD;

[Lee92] Julia C. Lee
A Transaction Management Model with Relational Database Operation Semantics
Ph.D Dissertation Northwestern University;
Evanston Dec, 1992

[Lyn83] Nancy A. Lynch;
Multilevel Atomicity-A new correctness Criterion for Database Concurrency Control;
ACM Transactions on Database Systems;

[MH89] William W. McCune and Lawrence J. Henschen
Maintaining State Constraints in Relational Database;
Journal of the Association for Computing Machinery;

[MRBKS92] Sharad Mehrotra, Rajeev Rastogi, Yuri Brietbart, Henry F. Korth,
Avi Silberschatz
The Concurrency Control Problem in Multidatabases: Characteristics and Solutions
Proceeding of the 1992 ACM SIGMOD;

[MS87] Hector Garcia-Molina, Kenneth Salem;
SAGAS;
Proceeding of SIGMOD;,
[Mal89] Carl Malamud;  
INGRES Tools for building an Information Architecture;  

[TSP92] John Turek, Dennis Shasha, Sundeep Praka  
Locking without Blocking - Making Lock Based Concurrent Data Structure Algorithms Nonblocking;  
Proceeding of Principles of Database Systems;  

[Ull88] Jeffrey D. Ullman;  
Chapter 9 Transaction Management;  
Database and Knowledge-base Systems;  

[WA92] M. H. Wong & D. Agrawal;  
Tolerating Bounded Inconsistency for Increasing Concurrency in Database Systems;  
Proceeding of ACM 11th Principles of Database Systems;  

[WT92] Paul Watson and Paul Townsend;  
The EDS Parallel Relational Database System;  
Parallel Database Systems;  

[Wei89] William E. Weihl;  
Local Atomicity Properties - Modular Concurrent Control for Abstract Data Types;  
ACM Transaction on Programming Language and Systems;  