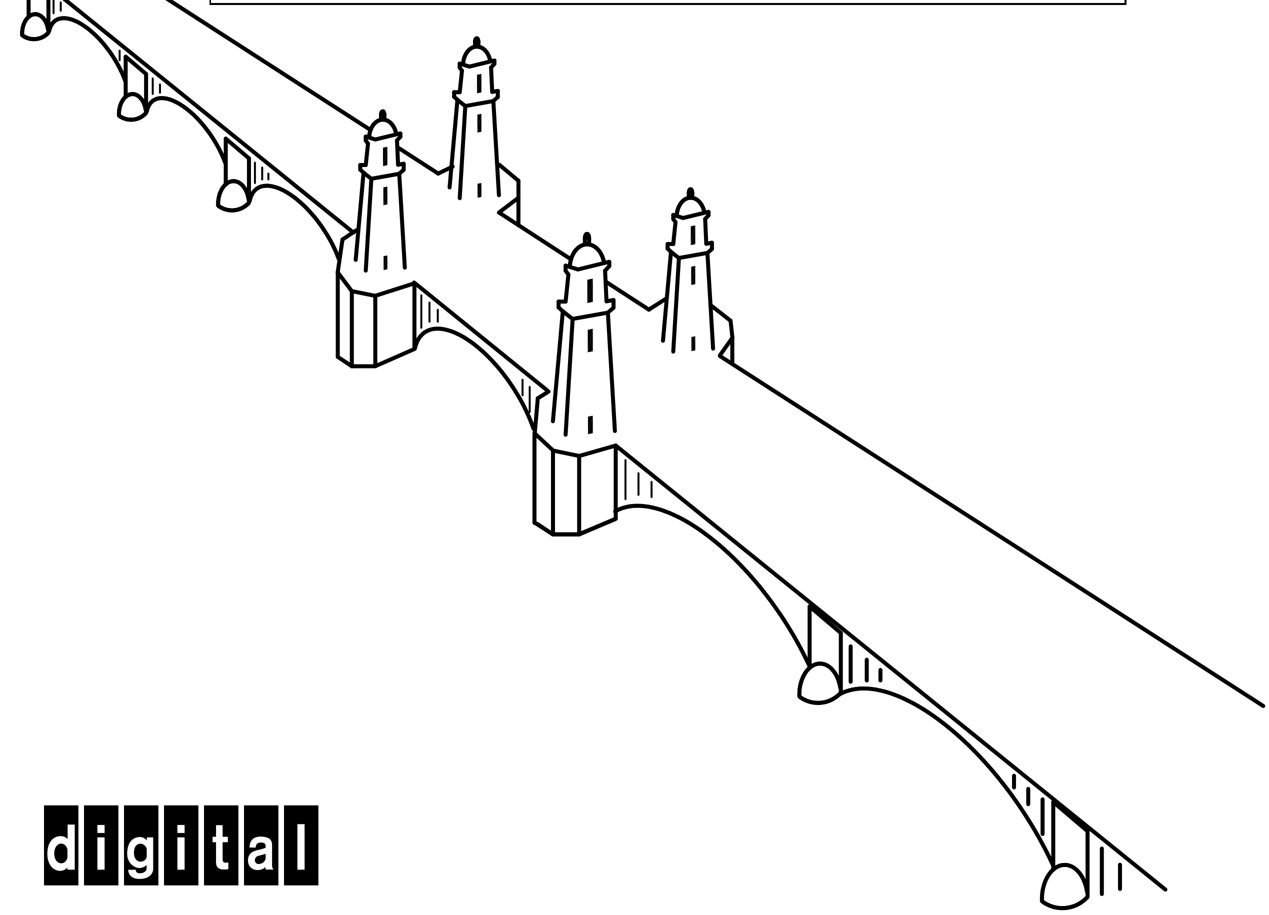
Key Range Locking Strategies for Improved Concurrency

David Lomet

Digital Equipment Corporation
Cambridge Research Lab

CRL 93/2 February 10, 1993



CAMBRIDGE RESEARCH LABORATORY
Technical Report Series

Digital Equipment Corporation has four research facilities: the Systems Research Center and the Western Research Laboratory, both in Palo Alto, California; the Paris Research Laboratory, in Paris; and the Cambridge Research Laboratory, in Cambridge, Massachusetts.

The Cambridge laboratory became operational in 1988 and is located at One Kendall Square, near MIT. CRL engages in computing research to extend the state of the computing art in areas likely to be important to Digital and its customers in future years. CRL's main focus is applications technology; that is, the creation of knowledge and tools useful for the preparation of important classes of applications.

CRL Technical Reports can be ordered by electronic mail. To receive instructions, send a message to one of the following addresses, with the word **help** in the Subject line:

On Digital's EASYnet:	CRL::TECHREPORTS
On the Internet:	techreports@crl.dec.com

digital™

Cambridge Research Laboratory
One Kendall Square
Cambridge, Massachusetts 02139

Key Range Locking Strategies for Improved Concurrency

David Lomet

Digital Equipment Corporation
Cambridge Research Lab

CRL 93/2 February 10, 1993

Abstract

"Phantoms" are records inserted into a concurrently selected set of records specified by a predicate such that the selection does not see a transaction consistent result. The usual locking strategy is to lock only existing records. However, records that satisfy the selection predicate must be locked even when they are not present in the database to prevent their insertion. One way to cope with phantoms is through range locking, a limited form of predicate locking. We investigate lock modes for ranges and describe new lock modes that greatly increase concurrency. We explore locking strategies involving range, key, and record resources that permit trade-offs between concurrency and lock overhead.

Keywords: concurrency, range locking, locking granularity, phantoms

1 Introduction

1.1 The Phantom Problem

Relational databases permit their users to select sets of tuples of relations based on a predicate. Transaction serializability requires that such a selected set not change until transaction completion. The implementation of a select normally relies on locking to preclude concurrent updates from changing the value of the set during the transaction.

Most lock manages are simple conflict detectors, blocking accesses to specific resources by detecting conflicting lock requests from different principals. So, the locking of specific records to protect them from change is straightforward. But, to keep a predicate-selected set from changing without materializing the set requires that insertions into a relation not satisfy outstanding selection predicates. Such inserted records are called phantoms. Preventing phantoms requires locks on records that do not exist at the time of the selection.

There are many solutions to the phantom problem. The simple ones involve locking a superset of the selected set. For example, one can lock a table whenever a subset of the table is selected. The goal has been to find a solution that has high concurrency and low lock overhead. Locking an entire table has low overhead but bad concurrency.

Predicate locking [2] is a complete solution to the phantom problem, but conventional lock managers (LMs) do not handle predicate locks. LMs need to have excellent performance and predicate locking is expensive and difficult. Index locking, with or without the index, has been suggested in [1]. The locking described there coped with phantoms when the predicates involved were equality predicates. Predicates that specify ranges of values were not handled.

1.2 Ranges as Lockable Granules

System R performed range locking within its B-tree indexing. This is credited to Eswaran and Blasgen in [4]. Recent papers [7, 8] have described this earlier work and have introduced improvements in both concurrency and lock overhead by showing how multigranularity lock modes can be exploited. Our work builds on these papers.

Because LMs are organized so as to detect conflicts only on specific named resources, key ranges must be uniquely named so that whenever we wish to operate on a record within the range, we "run into" the lock on the range that contains the record. This can be done by regarding index terms within indexes as defining ranges. The index terms then become the means to identify the ranges.

1.3 Forms of Resource Mapping

We deal with three resources, range, key and record. How these are identified, i.e., whether these are mapped to common or separate resource ids, can significantly affect the locking protocol, changing the trade-off between concurrency and locking overhead. We treat several alternatives for resource mapping.

1. range as a separate resource;

2. range and key identified as a common resource;

3. range, key and record identified as a common resource.

Locking overhead declines as the number of resources in the protocol declines. Interestingly, this can be done without loss of concurrency when ranges and keys are treated as common resources. Only when records are also "commoned" with ranges and keys is concurrency lost, but with a further reduction in locking overhead.

1.4 This Paper

We begin in section 2 by describing the locking when ranges are treated as separate from other resources and independent of key values. In section 3, we introduce key valued ranges, and show how to lock such ranges, first with the lock modes of multigranularity locking, and then via new lock modes. Section 4 describes, in two steps, how lock overhead is reduced by reducing the number of resources, and hence the number of locks. This section represents the heart of the paper and its main contribution. Starting with the new lock modes of the prior section, we show how to compose them to create additional lock modes such that no concurrency is lost when resources are merged. In this presentation, we consider the case where the records involved have unique keys. We conclude with a discussion in section 5, which includes a description of how non-unique keys might be handled.

2 Fundamentals

2.1 Multi-granularity Locking

Range locking is a restricted form of predicate locking. Multi-granularity locking(**MGL**) is a technique invented to cope with such preplanned predicates by regarding the predicates as resource "granules" that include or partially overlap with other resource granules [3]. MGL locking can be used for range locking.

MGL locking exploits lock modes in addition to the exclusion oriented S(share) and X(eXclusive) modes. These additional modes are known as "intention" modes. The purpose of an intention lock is to indicate, at a resource that contains other resources, that finer grained locking is permitted at the contained resources. An intention lock prevents another principal from acquiring a lock which permits it to access these other resources in a conflicting way without further locking.

Lock modes for MGL are given in Table 1. For example, IX (Intention eXclusive) conflicts with S. S enables access to contained resources in share mode without the need for additional locking, i.e., it implicitly locks (or covers) the contained resources. IX indicates that locking in X mode is occurring on contained resources. IX conflicts with S to block an S lock on the larger resource from implicitly S locking resources which are locked with explicit X locks. Similarly, IS indicates that locking in S mode is occurring on finer-grained resources.

2.2 Independent Ranges

Ranges of attribute values can be specified independently of the actual values of the attribute that are present. Such a specification is normally static, i.e., it does not vary with changes in the distribution of attribute values. With independent ranges, there is no need for an index on the file. For example, independent ranges can be defined via a fixed partitioning of the key space.

Lock Mode	IS	IX	S	SIX	X
IS	x	x	x	x	
IX	x	x			
S	x		x		
SIX	x				
X					

Table 1: The lock mode compatibility matrix for multi-granularity locking

We briefly treat independent ranges, but our primary purpose is to deal with key ranges. For fixed independent ranges, MGL solves the concurrency control problem. We can use either intention locks or covering locks (i.e. X or S locks) at the range level. We assume here that keys are unique.

2.2.1 Scans

For scans, one needs to lock ranges to prevent phantoms. The locks needed must be covering locks [5], i.e., locks that make it unnecessary to lock contained records since we have no way of explicitly locking each potential record of the range. Hence, scans need S locks on the ranges that they touch. An S range lock implicitly locks all records in the range. Explicit S record locks are not needed.

For a scan that will update some records, a range needs an SIX lock, with the updated records X locked. This implicitly locks all records of the range in S mode, but permits individual records of the range to be X locked. An update scan can avoid all record locking by X locking the range, at the cost of lost concurrency.

2.2.2 Singleton Record Operations

When modifying individual records, we can lock in one of two ways. Maximum concurrency is achieved by IX locking the range that contains the record being modified (updated, inserted or deleted), followed by X locking the record. Locking overhead is reduced if we X lock the range, hence "covering" contained records and avoiding the need to explicitly lock the

Operation	Range Lock	Key Lock
Read Single	none implicit IS	S
Update Single	IX	X
Read Scan	S	none required
Update Scan	SIX	none for read X for update
Insert	IX	X
Delete	IX	X

Table 2: Locking and lock modes required when ranges are resources independent of key values. In all cases, if prior operations of the transaction have locked a resource, the resulting lock must have a mode that covers both the new lock given in the table and the prior lock.

record itself. Concurrency is reduced when this is done, however, as only one modifier is now permitted within a range.

Single record reads can likewise be treated in two ways: Maximum concurrency is achieved by IS locking the range, followed by S locking the record. If one knows that X locks will not be set on a range, then one needn't post an IS lock on the range since it only serves to prevent X range locks. Hence, there need be no extra locking overhead for this approach to single record reads. The alternative is to always S lock the range that includes the record being read, avoiding the need for record locks.

2.2.3 Intention Locking Summary

The above is a direct application of multi-granularity locking. The summary table when intention locks are used for ranges when an individual record is read or written is given in Table 2. Using intention locks for ranges permits more potential concurrency, but also has more lock overhead, than does using covering locks. As noted, however, the extra cost need only arise when modifying data.

3 Separate Key Value Ranges

3.1 Key Valued Ranges

Consider an list of attribute values $(k_1, k_2, ...k_i)$, ordered such that $k_i < k_{i+1}$. This list defines disjoint semi-open ranges $(k_i, k_{i+1}]$. Each range can be identified by the value of its upper attribute. To lock the range $(k_i, k_{i+1}]$, we map k_{i+1} to a resource id and lock it. [4]. (Note that we could also define our ranges as $[k_i, k_{i+1})$ and identify them by k_i. To keep the ranges disjoint, one end must be open.)

Key ranges normally require the existence of an index so that the list of attribute values can be maintained in sort order. The indexed attribute name and the attribute value may be jointly used to name the ranges so as to distinguish like values of different attributes.

When distinguishing ranges from keys and records, we need two resource ids, one for the key k_i and one for the range $(k_{i-1}, k_i]$. Sometimes the resource id for the key value can be record identifier (rid) of the record containing the attribute value and hence whose rid is in the index term. The key value itself can then denote the range. In other cases, one might attach value and range tags to the key value. Other techniques are also possible.

Key value ranges differ from fixed ranges in that new attribute values can be inserted or old attribute values deleted, so the lockable ranges change over time. This keeps concurrency high because the ranges are small and adapt to the key distribution. The negative is the added complexity of splitting or consolidating ranges.

3.2 Permissible Concurrency

Permissible concurrency when ranges are distinct from records and keys is indicated in Table 3. The concurrency shown assumes the use of a conventional LM, and unique keys. Records are locked via their key values. Ranges are specified by the adjacent key values present in, e.g., an index. The result is that there is a single key value and hence record within each range. It is direct that two reads can be concurrent, but that updates should conflict with reads and other updates. The interesting part of Table 3 is the concurrency of deletes and inserts, and how they interact with reads and updates.

Insert and delete introduces new complexity because these operations

	Locked Items					
Operation	Read Record	Updated Record	Read Range	Updated Range	Insert Range	Delete Range
Read Record	Y	N	Y	N	N	Y^2
Update Record	N	N	N	N	N	$Y^{1,2}$
Read Scan	Y	N	Y	N	N	N
Update Scan						
(no record update)	Y	N	Y	N	N	N
(record updated)	N	N	N	N	N	N
Insert	$Y[ifo]^2$	$Y[ifo]^2$	N	N	Y	N
Delete	$Y[ifo]^2$	$Y[ifo]^2$	N	N	N	N

Table 3: The maximum concurrency for operations when confronted with locks of other transactions, when range and key locks are distinguished. A "Y" denotes concurrent execution, a "N" means exclusion is required. The "[ifo]" means "in front of" a read or updated record.
[1] Not achieved by with MGL modes and separate range and key resources.
[2] Not achieved with ARIES/KVL locking.

change the ranges identified by other key values as well as needing lock protection as operations. The interesting cases are:

Insert: Inserts can be permitted into a range formed by an uncommitted insert. The record that is inserted is itself locked, hence preventing its access while the inserter is active. However, inserts into a range $(k_{i-1}, k_{i+1}]$ with a delete cannot be permitted. If $k_{i'}$ were inserted, the lock on k_{i+1} to protect a deleted key k_i would cover only $(k_{i'}, k_{i+1}]$, and is ineffective if $k_{i'} > k_i$. Key range locks cannot conveniently handle this except by preventing all inserts into delete ranges.

Delete: Deletes into a range $(k_i, k_{i+1}]$ with an insert cannot be permitted. The delete needs to lock this range. But the inserted key k_{i+1} may be removed by a transaction abort, making locks on k_{i+1} invisible to subsequent operations, and hence leaving the range unguarded. Further, deletes in a delete range cannot be permitted. Were this to occur, and one of the deleting transactions subsequently aborted, an insert would occur into the delete range, dividing it. Above, we discussed why this

Operation	Current Key Range	Key	Next Key Range
Read Single	IS	S	
Update Single	IX	X	
Read Scan	S	none	
Update Scan	S (on read) SIX (on modify)	none X	
Insert	IX SIX if next range locked with S or SIX	X	IX (instant)
Delete	SIX (instant)	X	SIX

Table 4: Locking required when key valued locks are used to lock ranges that are separate from keys. Each range has one record and range boundaries can change by insertion or deletion of key values. This locking exploits existing MGL lock modes. When resources are already held by the transaction, the resulting lock needs to cover both previous and new locks.

cannot be permitted.

Read and Update: Operations on records whose keys denote delete ranges can continue when the records are not involved in prior operations. This is not possible for insert ranges where the key that denotes a range is the inserted key. Inserts and deletes in front of read or updated records are permissible.

3.3 Inserts and Deletes Using MGL Lock Modes

The locking for key valued ranges when MGL lock modes are used is indicated in Table 4. The locking of Table 2 is little changed. The ranges are now named by key values. A range only contains a single record. Thus Read Single, Update Single, and Read Scan are essentially unchanged. For Update Scan, only one record appears in each range, and it is either updated or read. If only read, then our range lock can be S, not SIX.

3.3.1 Insert

The insert of k_i divides range $(k_{i-1}, k_{i+1}]$ into two ranges, $(k_{i-1}, k_i]$ and $(k_i, k_{i+1}]$. The following steps are needed:

1. An instant IX lock (the "Next Range Lock" in Table 4) checks whether range $(k_{i-1}, k_{i+1}]$ has a conflicting lock, e.g., S or SIX. If not then key k_i is inserted. The lock on $(k_{i-1}, k_{i+1}]$ can be instant because (i) it tests that the range is not locked; (ii) this range is then divided; and (iii) after division, the range protected by a lock involving k_{i+1} is $k_i, k_{i+1}]$, which does not guard any activity and does not need protection.

2. $(k_{i-1}, k_i]$ is then locked in IX (for commit duration); and

3. the $k_{i'}$ key value is X locked (for commit duration);

4. if the $(k_{i-1}, k_{i+1}]$ range had been previously locked by this transaction, in general, one does not know which of the resulting ranges still needs the protecting. Thus, if the current transaction held an S or SIX lock on this next range, we propagate the effect of this lock to the new range $(k_{i-1}, k_i]$. This requires an SIX lock regardless of which of S or SIX were held previously on the original range, as SIX is the least upper bound for (SIX, S) and IX.

3.3.2 Delete

The delete of k_i merges ranges $(k_{i-1}, k_i]$ and $(k_i, k_{i+1}]$. We use SIX to guard the merged range (called a delete range) $(k_{i-1}, k_{i+1}]$ that results. The following steps are needed:

1. Delete checks whether the k_i record can be deleted with an X lock on the k_i record, of commit duration to prevent insertion of another k_i record.

2. The $(k_{i-1}, k_i]$ range is locked with SIX to ensure that a scan does not prevent the delete. This is an instant lock since k_i is being removed and hence will not show up for subsequent locking.

3. The merged range $(k_{i-1}, k_{i+1}]$ is SIX locked for commit duration to guard the delete, preventing insertions and scans.

Using existing locks prevents some activity that is permissible. We guard a delete range $(k_{i-1}, k_{i+1}]$ using SIX. Unfortunately, since SIX conflicts with IX, it prevents record k_{i+1} from being updated, even when the key is not locked.

3.3.3 ARIES/KVL and ARIES/IM

By distinguishing range from keys, we gain extra concurrency compared with ARIES/KVL [7] or ARIES/IM [8], which also use existing lock modes but do not distinguish ranges from keys. In particular, we can use an SIX lock for delete ranges instead of an X lock. Our cost is extra overhead, sometimes needing key locks as well as range locks.

Below, we further improve concurrency by introducing new lock modes. However, the lock overhead still remains high. In section 4, we merge range and key resources and, using new lock modes, gain maximal key range concurrency with lock overhead comparable with ARIES/KVL. Further resource commoning reduces concurrency but still gives better concurrency than ARIES/IM and with comparable lock overhead.

3.4 New Lock Modes for Changing Ranges

The problem with using existing lock modes for range locking is that they do not distinguish with sufficient refinement the nature of the operations that are being performed, particularly those that involve changes which are indicated by an IX mode lock. Here we introduce three lock modes to replace IX, one mode for each form of modification operation.

We guard a range with the form of intention lock that reflects the operation that has occurred within the range. It is held until commit to prevent subsequent conflicting operations in the range. While we find it necessary for intention modes to reflect a specific modify operation, it is sufficient for the key lock to simply indicate that exclusion is needed via an X mode lock. We lose no concurrency by doing this. Thus we have the following:

Intention Update(IU): an IU lock indicates that the record forming the upper bound of a range is being updated.

Intention Insert(IIn): an IIn lock indicates that the record forming the upper bound of a range is the result of an insertion.

Operation	Range Lock	Key Lock	Next Range Lock
Update	IU	X	
Insert	IIn ID if next range locked ID SIX if next range locked S or SIX	X	IIn (instant)
Delete	ID (instant)	X	ID

Table 5: Locking required for updates, and for insert and delete operations that cause ranges to change, when key valued locks are used and ranges are distinct from keys. The locking exploits new lock modes. Once again, if the resource is already locked by the transaction, the new lock must cover both previous and new locks.

Intention Delete(ID): an ID lock indicates that the range identified by the record forming its upper bound contains a deletion.

These new lock modes permit us to avoid using the overly restrictive SIX lock to guard delete ranges. Locking for update, insert, and delete are illustrated in Table 5. Read and scan locking do not change.

The new intention modes behave like the IX mode with respect to other lock modes. However, their compatibilities among themselves differ so as to facilitate increased concurrency for dynamically changing ranges. Their compatibilities are described in Table 6.

The new lock mode compatibilities permit us to maximize the concurrency that is possible within the general framework of key valued locks (see Table 3). In particular, we can now update a record whose key guards a delete range. (While this is a small improvement, we subsequently use these lock modes in more substantive ways.) What we have done is very straightforward. We made the intention lock modes operation specific. Then we constructed the lock mode compatibility table to directly correspond to the exclusion and compatibility needed by each operation.

Once again, we need to handle ranges already locked by our transaction when range boundaries change. If the range $(k_{i-1}, k_{i+1}]$ was previously scanned, when that range is divided by the insertion of k_i, the range $(k_{i-1}, k_i]$ must remain scan locked and SIX is sufficient for this. If this range guards

Lock Mode	IS	IU	IIn	ID	S	SIX	X
IS	x	x	x	x	x	x	
IU	x	x	x	x			
IIn	x	x	x				
ID	x	x					
S	x				x		
SIX	x						
X							

Table 6: The compatibilities of lock modes, including now the new intention locks for guarding ranges in which a change has occurred. These are more restrictive than IX (except for IU) but less restrictive than SIX.

a delete, then the new current range must similarly guard a delete with an ID lock. For deletes, if the range $(k_i, k_{i+1}]$ has been scanned, this range has now been scanned *and* guards a delete. An SIX lock is sufficient for this.

4 Common Resources for Reduced Locking

4.1 Ranges and Keys in Common

Our goal is to achieve the highest possible concurrency with the lowest possible locking overhead. Concurrency is assessed by determining how many operations are prevented unnecessarily by locks that are too "heavy handed". Lock overhead is measured by the number of locks needed by the locking protocol. We first discuss range locking using existing MGL locks, then introduce new lock modes to achieve greater concurrency. Here, the key value identifies both itself and the range for which it is an upper bound.

4.1.1 Key Value Locks(KVL)

The best locking protocol for range locking using existing MGL lock modes is ARIES/KVL [7]. It does not distinguish range from key lock modes. usually locks with modes which are the least upper bound of the range and key lock

Operation	Current Key	Next Key
Read Single	S	
Update Single	X	
Read Scan Update Scan	S X	
Insert	IX X if next key is locked with S, SIX, X	IX (instant)
Delete	X (instant)	X

Table 7: Locking required in ARIES/KVL for the various operations. ARIES/KVL identifies ranges and keys as the same resource and uses MGL lock modes.

modes on the separate resources are used. Sometimes the stronger lock mode for one operation will be sufficient to protect against the conflicts that must be precluded without strengthening another operation's lock mode. For the operations that we are considering, the ARIES/KVL lock table is given in Table 7 (for unique keys).

The first four operations simply use the stronger key lock as the range lock. Insert uses the weaker IX lock. This is satisfactory in preventing read, update or delete of the inserted record since IX conflicts with S and X. Since IX is compatible with IX, inserts in front of inserts are permitted. For deletes, the X next key lock prevents inserts into a delete range.

The result of folding range and record into a single resource is to prevent inserts in front of Read Single and Update Single. The X lock for the key guarding the delete range prevents the reading or update of this record, even though the record may not itself be the subject of conflicting operations. It also prevents deletes in front of such records. (See Table 3.) Thus, concurrency is impeded where logically the operations do not conflict.

4.1.2 Expanded Lock Modes(KRL)

It is possible to achieve high concurrency by expanding the lock modes once again so that the lock modes denote the separate lock modes needed by both

Operation	Current Key	Next Key
Read Single	IS-S	
Update Single	IU-X	
Read Scan	S	
Update Scan	S (on read) X (on modify)	
Insert	IIn-X X if next key is locked with ID-,S,SIX,X	IIn- (instant)
Delete	X (instant)	ID-

Table 8: Locking and lock modes required when KRL locks for both ranges and records are mapped into a single resource.

range and key. Thus, our lock modes are pairs $(RangeMode, KeyMode)$. The $RangeMode$ can be an intention lock or a covering lock, while $KeyMode$ may be any simple covering lock or the null lock. For example, a lock mode of $ID - X$ on a resource denoted by k_i means that the range $(k_{i-1}, k_i]$ is locked in mode ID while the key k_i is locked in mode X. We call our locks key range locks or **KRL** locks.

The lock modes for our operations are given in Table 8. There are only eight distinct lock modes while the cartesian product of lock modes for ranges and lock modes for keys is 18, six range lock modes $(IS, IU, IIn, ID, S, SIX)$ times three key lock modes $(null, S, X)$. The combinations $(IS-, IS - X, IU-, IU - S, IIn - S, S - S, S - X, SIX - S)$ do not arise. Some combinations share compatibilities: $(ID - X, SIX - X)$ denoted as X since it is incompatible with all other modes, and $(ID - S, SIX-)$ denoted as SIX since it is compatible only with an intention shared lock mode, here $IS - S$. We rename $S-$ as S since its compatibilities are the same.

The lock mode compatibility for our new lock modes is defined in Table 9. It is a direct result of composing the separate lock compatibilities of ranges and records. That is, $(RangeMode_1, KeyMode_1)$ is compatible with $(RangeMode_2, KeyMode_2)$ if $RangeMode_1$ is compatible with $RangeMode_2$ and $KeyMode_1$ is compatible with $KeyMode_2$.

MODES	IS-S	IIn-	ID-	IU-X	IIn-X	S	SIX	X
IS-S	x	x	x			x	x	
IIn-	x	x		x	x			
ID-	x			x				
IU-X		x	x					
IIn-X		x						
S	x					x		
SIX	x							
X								

Table 9: The lock mode compatibility matrix for the new combined resource lock modes needed for KRL locking.

The concurrency permitted when using separate range and key locks is achieved using KRL lock modes on a single combined (*range*, *key*) resource. Where separate range and key locks required that both range and key locks be acquired, the KRL protocol requires that one lock be acquired that satisfies the constraints of both these separate locks. KRL distinguishes precisely the separate range and record conflicts by its composite lock mode. ARIES/KVL locking, with its restriction to traditional lock modes, does not.

As with ARIES/KVL locking, KRL locking permits or precludes the following.

1. Inserts in front of inserts are still permitted as the $IIn-$ instant lock is compatible with the $IIn-X$ commit duration lock.

2. Changes to a range that has been scanned are precluded as no X lock of any kind is permitted for either S, SIX, or X.

3. No inserts into a delete range are possible because $IIn-$ conflicts with $ID-$, SIX, and X.

4. Deletes in front of inserts are precluded because $ID-$ conflicts with $IIn-X$.

5. Multiple scans, including update scans, are possible so long as there is no record updated because S locks are compatible. Similarly, singleton

reads are permitted in scanned ranges, so long as the record involved has not been updated, because $IS - S$ is compatible with S and SIX.

6. Scans into delete guarded ranges are precluded because $ID-$ conflicts with SIX, S, and X.

The following are correctly permitted by KRL locking, but are precluded by ARIES/KVL locking.

1. Inserts in front of singleton reads and updates are permitted because $IIn-$ is compatible with $IS - S$ and $IU - X$.

2. Deletes in front of singleton reads and updates are permitted because $ID-$ is compatible with $IS-S$ and $IU-X$, which also permits Reading or updating a record guarding a delete range.

KRL locking achieves *all* of the concurrency indicated as possible in Table 3.

4.2 Ranges, Keys and Records in Common

4.2.1 Multiple Paths and Locking

Multi-granularity locking can be effective not only in tree resource hierarchies but also for directed acyclic graphs(DAGs). However, when the resource hierarchy is a DAG, our locking protocol must provide an appropriate level of exclusion on all paths to the underlying data.

A KRL lock applied to a common $(range, key)$ resource locks only the access path that proceeds by way of the index on the key (attribute) in question. This provides for high concurrency because it permits accesses on other paths to proceed. Thus, this approach distinguishes and locks only the ranges that have been scanned (see Figure 1(a)). However, operations on specific records require that exclusion by applied on all paths. For example, we must preclude other operations on an updated record, regardless of which access path is traversed to the record.

For DAG locking via key value locks to be effective, different key values may need to be locked in several indexes, e.g. for insert and delete operations, all indexed values need updating and hence locking. The reading of a record via a scan in one index needs to be visible as a singleton read via other

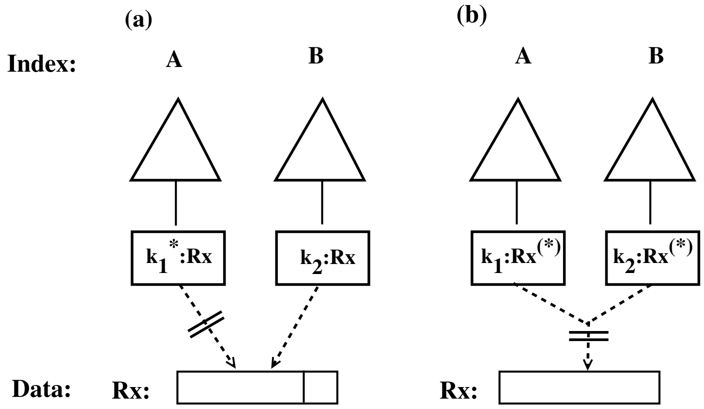


Figure 1: Locking key values is contrasted with locking records for locking ranges. (a) Locking a range and key on a path, here from index A, permits accesses through other indexes, here index B. (b) A record lock used as a range lock locks all paths that involve attributes from the record, here paths from indexes A and B. It also locks the access path that bypasses all indexes.

indexes. Also, records are usually accessible on an access path that does not traverse an index. This access path needs lock protection as well.

The bottom line here is that, while concurrency is high because the ranges denoted by the key values in each index are carefully discriminated and kept separate, the cost in lock overhead and complexity can be high. One solution is to acquire record locks as well as key/range locks in an index.

4.2.2 Record Locks

One can shortcircuit the additional locking described above by treating the lock needed for range, key, **and** record as a single resource, as proposed in ARIES/IM [8], described briefly below. The penalty of doing such consolidation of resources is less concurrency. Can KRL locking be applied when this is done? The answer is yes.

ARIES/IM uses a lock on a record identifier instead of a key value lock as the way to lock a range. This one lock then serves as range, key and record lock. An index entry consists of a pair $< key, rid >$. When a scan is done through an index, range, key value and record resources are all identified by a single rid. This is a great simplification and reduces locking overhead, but at the expense of concurrency. The locked rid locks all ranges which contain attribute values from the record. Essentially we are preventing "phantoms" in key value ranges that do not participate in the scan. This is illustrated in Figure 1(b).

We cannot easily escape from the reduced concurrency of record locking with the invention of additional lock modes. The problem is that the lock modes for the common resource denoted by rid do not distinguish which ranges we really intended to lock. And the number of ranges involved varies with the database design, and can be quite large in any event. Thus, inventing new lock modes does not seem to be an effective strategy.

We can, however, combine KRL lock modes with the record locking technique for range locks. This achieves greater concurrency than ARIES/IM. For example, KRL successfully distinguishes record locks for singleton updating and reading from range locks for phantom protection. ARIES/IM does not. However, we can no longer maintain that the locking is optimal. Each range lock on a record will serve to lock all ranges that involve attributes values of the record.

When the data of a file is stored and clustered via a primary index, the primary key becomes the record identifier. Hence, key locking in a primary index is the same as record locking via an rid. Scans that involve the primary key then have the low overhead of the ARIES/IM's record locking approach, even when doing key valued locking.

5 Discussion

Three other issues are worthy of some discussion.

5.1 Scans Directly on Records

Not all scans are through indexes. Not all attributes are indexed, and not all predicates controlling a scan can make effective use of an index. For these scans, one needs to search through an entire table. This is, in fact, very little different whether the table is clustered by primary key or by a more arbitrary rid. Table scans still need to guard against phantoms. So long as records might be inserted or deleted in the midst of already scanned records, these operations can give rise to phantoms.

KRL locks are effective in preventing phantoms for these clustering order scans. A record and the range between it and its immediate *physical* predecessor is locked during a scan via the record's rid. The range locks do not protect a specific logical predicate or key value range. Rather, the locks prevent insertions into physical ranges that have been previously scanned by active transactions.

5.2 Non-unique Keys

Non-unique keys occur when an attribute value can occur more than once. An index on hair color would find many people with brown hair. We need to look again at locking protocols, their lock overhead, and the concurrency that they can achieve.

One approach works well when rids are used to identify ranges and keys to the lock manager. We organize the entries of an index in sort order by $< key, rid >$ pair. These *pairs* are unique entries. Insertion of a $< key, rid >$ is handled as a unique key insertion. The new entry, its key and rid, is inserted into this ordered list. This does not optimize the potential concurrency, as the ordering of rids imposes a logically unnecessary restriction on where a new entry might be inserted. But the concurrency is good and the KRL locking protocol is unchanged. This technique does not require that each $< key, rid >$ be stored as a physically separate entry. One can have an index where an entry consists of a key paired with the ordered list of rids. It simply means that rid order must be preserved.

More concurrency is possible if the entries above are not required to be ordered by rid within the key ordering. Then, one might, for example, insert a new rid into a set of rids associated with a key anywhere that was convenient. The result, however, is a more complex protocol. ARIES/KVL [7] discusses the non-unique key case when keys, not records, are the lockable resources, and illustrates the less restrictive locking that might be exploited. The concurrency advantages of the ARIES/KVL approach versus ordering entries by $< key, rid >$ and using KRL locking are not clear. While ARIES/KVL does not impose an rid ordering, and hence will sometimes permit additional concurrency, an ARIES/KVL key value lock in a scan locks all the records with that key value. This lowers locking overhead but leads to less concurrency.

5.3 Deletions as Updates

If deleted keys or records were marked as deleted, instead of being removed, then some of the strict exclusion required by deletions can be relaxed. Deletion could then be treated as a record update. Actual removal of the "delete stub" could be done in a separate short atomic action when this stub is not locked. This improves concurrency as marking deletes does not consolidate ranges. It does, however, increase storage costs and requires a subsequent garbage collection step.

5.4 Summary

Our range locking protocols work with a conventional LM, in which the LM is a simple conflict detector for locks on discrete resources. Such an LM does not know about dependencies between resources, and, in particular does not work directly with ranges. Within this conventional context, and using key value ranges based on actual keys present, we derived Table 3. It describes the maximum concurrency possible under these conditions. We then introduced new lock modes to cope with insert and delete operations, which can change the ranges described by key values and showed that these lock modes achieve this maximum concurrency.

To reduce lock overhead, we unified range and key resources so that both can be locked with a single lock. This reduces the locking overhead dramatically. Maximum concurrency was retained when we introduced KRL compound lock modes that distinguish the separate roles of ranges and keys, but with greatly reduced locking overhead.

Finally, to even further reduce lock overhead, we combined range, key, and record into a single lockable resource, a technique exploited in ARIES/IM. This reduces concurrency because one record lock can lock ranges for multiple attribute (key) ranges, even when the additional ranges do not require locking. This appears to be an intrinsic penalty of this mapping. KRL lock modes do very well under these conditions, strictly better than previous solutions. The extraneously locked ranges prevent us from claiming optimality.

References

[1] Bernstein, P., Hadzilacos, V. and Goodman, N. *Concurrency Control and Recovery in Database Systems* Addison Wesley, Reading MA (1987)

[2] Eswaren, K., Gray, J., Lorie, R., and Traiger, I. The notions of consistency and predicate locks in a database system. *Communications of the ACM* 19,11 (Nov. 1976) 624-633.

[3] Gray, J.N., Lorie, R. A., Putzulo, G. R., and Traiger, I. L. Granularity of locks and degrees of consistency in a shared data base. *IFIP Working Conf on Modeling of Data Base Management Systems* (1976) 1-29.

[4] Gray, J. and Reuter, A. *Transaction Processing: Concepts and Techniques.* Morgan Kaufmann, San Mateo, CA (1992)

[5] Lomet, D. Private Lock Management Digital Equipment Corp. Tech Report (draft) (Aug. 1992) Cambridge Research Lab, Cambridge, MA.

[6] Lomet, D. and Green, R. Key range locking with index trees. Digital Equipment Corp. Tech Report (draft) (Sept. 1992) Cambridge Research Lab, Cambridge, MA.

[7] Mohan, C. ARIES/KVL: A key-value locking method for concurrency control of multiaction transactions operating on B-tree indexes, *Proc. Very Large Databases Conf.* Brisbane, AU (Aug 1990).

[8] Mohan, C. and Levine, F. ARIES/IM: an efficient and high concurrency index management method using write-ahead logging. IBM Research Report RJ 6846, (Aug 1989) Almaden Research Center, San Jose, CA.

Contents