Transactional Information Systems:

Theory, Algorithms, and the Practice of Concurrency Control and Recovery

Gerhard Weikum and Gottfried Vossen

© 2002 Morgan Kaufmann
ISBN 1-55860-508-8

“Teamwork is essential. It allows you to blame someone else.” (Anonymous)
Part II: Concurrency Control

• 3 Concurrency Control: Notions of Correctness for the Page Model
• 4 Concurrency Control Algorithms
• 5 Multiversion Concurrency Control
• 6 Concurrency Control on Objects: Notions of Correctness
• 7 Concurrency Control Algorithms on Objects
• 8 Concurrency Control on Relational Databases
• 9 Concurrency Control on Search Structures
• 10 Implementation and Pragmatic Issues
7 Concurrency Control
Algorithms on Objects

• 7.2 Locking for Flat Object Transactions
  • 7.3 Layered Locking
  • 7.4 Locking on General Transaction Forests
  • 7.5 Hybrid Algorithms
  • 7.6 Locking for Return-value Commutativity
  • 7.7 Lessons Learned

“A journey of thousand miles must begin with a single step.” (Lao-tse)
2PL for Flat Object Schedules

- introduce a special lock mode for each operation type
- derive lock compatibility from state-independent commutativity

**Lock acquisition rule:**
L₁ operation f(x) needs to lock x in f mode

**Lock release rule:**
Once an L₁ lock of f(x) is released, no other L₁ lock can be acquired.

**Example:**

```
<table>
<thead>
<tr>
<th></th>
<th>deposit(a)</th>
<th>deposit(b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>t₁</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>withdraw (c)</td>
<td>withdraw(a)</td>
</tr>
<tr>
<td>t₂</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
7 Concurrency Control
Algorithms on Objects

- 7.2 Locking for Flat Object Transactions
- **7.3 Layered Locking**
  - 7.4 Locking on General Transaction Forests
  - 7.5 Hybrid Algorithms
  - 7.6 Locking for Return-value Commutativity
  - 7.7 Lessons Learned
Layered 2PL

- **Lock acquisition rule:**
  \( \text{L}_i \) operation \( f(x) \) with parent \( p \), which is now a subtransaction, needs to lock \( x \) in \( f \) mode

- **Lock release rule:**
  Once an \( \text{L}_i \) lock of \( f(x) \) with parent \( p \) is released, no other child of \( p \) can acquire any locks.

- **Subtransaction rule:**
  At the termination of an \( \text{L}_i \) operation \( f(x) \), all \( \text{L}_{(i-1)} \) locks acquired for children of \( f(x) \) are released.

**Theorem 7.1:**
Layered 2PL generates only tree reducible schedules.

**Proof:** All level-to-level schedules are OCSR, hence the claim (by Theorem 6.2).

**Special cases:**
- single-page subtransactions merely need **latching**
- for all-commutative \( \text{L}_i \) operations, transactions are decomposed into sequences of independently isolated, **chained subtransactions**
2-Level 2PL Example

- fetch(x)
- modify(y)
- modify(w)
- store(z)
- r(t)
- r(p)
- w(q)
- w(p)
- w(t)

Time points:
- $t_1$
- $t_2$
- $t_{11}$
- $t_{12}$
- $t_{13}$
- $t_{21}$
- $t_{22}$

Levels:
- $L_0$
- $L_1$
3-Level Example

Insert Into Persons
Values (Name=..., City="Austin", Age=29, ...)

Select Name
From Persons
Where City="Seattle"
And Age=29

Select Name
From Persons
Where Age=30

store(x)
insert
(CityIndex, "Austin", @x)

search
(CityIndex, "Seattle")

insert
(AgeIndex, 29, @x)

search
(AgeIndex, 29)

search
(AgeIndex, 30)

fetch(y)

r(p)
w(p)
r(r)
r(n)

r(r) r(l) r(n) w(l) r(l) r(r) r(n) r(l) r(r) r(n) r(l) w(l) r(p) w(p)

fetch(z)

r(r) r(n) r(l) r(p) w(p)
3-Level 2PL Example

Insert Into Persons
Values (Name=..., City="Austin", Age=29, ...)

Select Name From Persons
Where City="Seattle" And Age=29

Select Name From Persons
Where Age=30

store(x)
insert (CityIndex, "Austin", @x)
search (CityIndex, "Seattle")
insert (AgeIndex, 29, @x)
search (AgeIndex, 30)
fetch(z)

r(p) w(p)
r(r) r(n) r(l) w(l)

r(r"r(n")r(l")w(l")

r(r"r(n")r(l")
r(p) w(p)

r(r)p w(p)
r(r) r(n) r(l)

r(r")r(n")r(l")
r(p) w(p)
Selective Layered 2PL

For n-level schedule with layers $L_n, ..., L_0$
apply locking on selected layers $L_{i_0}, ..., L_{i_k}$
with $1 \leq k \leq n$, $i_0 = n$, $i_k = 0$, $i_\nu > i_{\nu + 1}$,
skipping all other layers

- **Lock acquisition rule:**
  $L_i$ operation $f(x)$ with $L_{i-1}$ ancestor $p$, which is now a subtransaction,
  needs to lock $x$ in $f$ mode

- **Lock release rule:**
  Once an $L_i$ lock of $f(x)$ with $L_{i-1}$ ancestor $p$ is released,
  no other $L_i$ descendant of $p$ can acquire any locks.

- **Subtransaction rule:**
  At the termination of an $L_i$ operation $f(x)$,
  all $L_{i+1}$ locks acquired for descendants of $f(x)$ are released.
Selective Layered 2PL Example

Insert Into Persons
Values (Name=..., City="Austin", Age=29, ...)

Select Name From Persons
Where City="Seattle" And Age=29

Select Name From Persons
Where Age=30

store(x)

insert (CityIndex, "Austin", @x)

search (CityIndex, "Seattle")

insert (AgeIndex, 29, @x)

search (AgeIndex, 30)

fetch(z)

r(p)

w(p)

r(r) r(n) r(l) w(l)

r(r") r(n") r(l") w(l")
7 Concurrency Control
Algorithms on Objects

• 7.2 Locking for Flat Object Transactions
• 7.3 Layered Locking
• 7.4 Locking on General Transaction Forests
• 7.5 Hybrid Algorithms
• 7.6 Locking for Return-value Commutativity
• 7.7 Lessons Learned
Problem Scenario

Problem: layers can be “bypassed”
Solution: keep locks in “retained” mode
General Object-Model 2PL

- **Lock acquisition rule:**
  Operation \( f(x) \) with parent \( p \) needs to lock \( x \) in \( f \) mode

- **Lock conflict rule:**
  A lock requested by \( r(x) \) is granted if
  - either no conflicting lock on \( x \) is held
  - or when for every transaction that holds a conflicting lock, say \( h(x) \),
    \( h(x) \) is a retained lock and \( r \) and \( h \) have ancestors \( r' \) and \( h' \) such that
    \( h' \) is terminated and commutes with \( r' \)

- **Lock release rule:**
  Once a lock of \( f(x) \) with parent \( p \) is released,
  no other child of \( p \) can acquire any locks.

- **Subtransaction rule:**
  At the termination of \( f(x) \),
  all locks acquired for children of \( f(x) \) are converted into retained locks.

- **Transaction rule:**
  At the termination of a transaction, all locks are released.

**Theorem 7.2:**
The object-model 2PL generates only tree-reducible schedules.
Proof Sketch for Theorem 7.2

- If all locks of $t_1$ were kept until commit, then tree reducibility were trivially guaranteed.
- Now show that retained $f_1$ lock by $h_1$ is sufficient to prevent non-commutative subtree:

  Let $f_2$ be the first conflict with any lock under $h_1$; $f_2$ is allowed to proceed only if $h_1$ is terminated and $h_2$ commutes with $h_1$.
  - $\rightarrow$ isolate $h_2$ from $h_1$
  - $\rightarrow$ prune $h_2$ and $h_1$
  - $\rightarrow$ commute $h_2$ with $h_1$ if necessary
7 Concurrency Control
Algorithms on Objects

• 7.2 Locking for Flat Object Transactions
• 7.3 Layered Locking
• 7.4 Locking on General Transaction Forests

• 7.5 Hybrid Algorithms
• 7.6 Locking for Return-value Commutativity
• 7.7 Lessons Learned
Hybrid Algorithms

**Theorem 7.3:**
For 2-level schedules the combination of 2PL at L₁ and FOCC at L₀ generates only tree-reducible schedules.

**Theorem 7.4:**
For 2-level schedules the combination of 2PL at L₁ and ROMV at L₀ generates only tree-reducible schedules.

*These combinations are particularly attractive because subtransactions are short and there is a large fraction of read-only subtransactions.*
7 Concurrency Control
Algorithms on Objects

• 7.2 Locking for Flat Object Transactions
• 7.3 Layered Locking
• 7.4 Locking on General Transaction Forests
• 7.5 Hybrid Algorithms
• 7.6 Locking for Return-value Commutativity
• 7.7 Lessons Learned
Locking for Return-value Commutativity

• introduce a special lock mode for each pair <operation type, return value>,
  Example: lock modes
  withdraw-ok, withdraw-no, deposit-ok, getbalance-ok
• defer lock conflict test until end of subtransaction
• rollback subtransaction if lock cannot be granted and retry
Escrow Locking

... on bounded counter object \( x \) with
lower bound \( \text{low}(x) \) and upper bound \( \text{high}(x) \)

**Approach:**
- maintain infimum \( \text{inf}(x) \) and supremum \( \text{sup}(x) \) for the value of \( x \)
taking into account all possible outcomes of active transactions
- adjust \( \text{inf}(x) \) and \( \text{sup}(x) \) upon
  - operations \( \text{incr}(x), \text{decr}(x) \), and
  - commit or abort of transactions
## Escrow Locking Pseudocode

### incr(x, Δ):
- if \( x.\sup + \Delta \leq x.\text{high} \) then
  - \( x.\sup := x.\sup + \Delta \); return ok
- else if \( x.\inf + \Delta > x.\text{high} \) then
  - return no
- else wait fi fi;

### decr(x, Δ):
- if \( x.\text{low} \leq x.\inf - \Delta \) then
  - \( x.\inf := x.\inf - \Delta \); return ok
- else if \( x.\text{low} > x.\sup - \Delta \) then
  - return no
- else wait fi fi;

### commit(t):
- for each op incr(x, Δ) by t do
  - \( x.\inf := x.\inf + \Delta \) od;
- for each op decr(x, Δ) by t do
  - \( x.\sup := x.\sup - \Delta \) od;

### abort(t):
- for each op incr(x, Δ) by t do
  - \( x.\sup := x.\sup - \Delta \) od;
- for each op decr(x, Δ) by t do
  - \( x.\inf := x.\inf + \Delta \) od;
Escrow Locking Example

\[ x^{(0)} = 100 \]

constraint:
\[ 0 \leq x \]

\[ x^{(4)} = 50 \]
Escrow Deadlock Example

\[ x^{(0)} = 0 \]

\[ \text{incr}(x, 10) \quad \text{update}(y) \]

\[ \text{incr}(x, 10) \quad \text{update}(z) \]

\[ \text{incr}(x, 10) \]

\[ \text{getval}(y) \quad \text{getval}(z) \quad \text{decr}(x, 20) \]
7 Concurrency Control
Algorithms on Objects

• 7.2 Locking for Flat Object Transactions
• 7.3 Layered Locking
• 7.4 Locking on General Transaction Forests
• 7.5 Hybrid Algorithms
• 7.6 Locking for Return-value Commutativity
• 7.7 Lessons Learned
Lessons Learned

• Layered 2PL is the fundamental protocol for industrial-strength data servers with record granularity locking (it explains the trick of “long locking” and “short latching”).
• This works for all kinds of ADT operations within layers; decomposed transactions with chained subtransactions (aka. “Sagas”) are simply a special case.
• Non-layered schedules require additional, careful locking rules.
• Locking on some layers can be combined with other protocols (e.g., ROMV or FOCC) on other layers.
• Escrow locking on counter objects is an example for additional performance enhancements by exploiting rv commutativity.