

On Optimistic Methods for Concurrency Control

Neeraj Gupta

Additional Source: Database Management Systems – Ramakrishnan, Gehrke

10/07/09

EECS 584, Fall 2009

1

Contents

- Introduction to concurrency control
- Phases of Optimistic Concurrency Control
- Practical Considerations
- Applications

10/07/09

EECS 584, Fall 2009

2

Concurrency Control

- Why ?
 - To maximize throughput
 - Hardware is underutilized on low concurrency degree
- How to guarantee ?
 - Pessimistic Concurrency Control (using locks)
 - Optimistic Concurrency Control methods

10/07/09

EECS 584, Fall 2009

3

Pessimistic Concurrency Control

- Assumes conflict will happen
- Disadvantages:
 - Lock maintenance overhead
 - Possibility of deadlocks
 - Concurrency significantly lowered
 - When congested nodes are locked
 - To allow transaction abort
 - Sometimes, conflicts are rare

10/04/09

EECS 584, Fall 2009

4

Optimistic Concurrency Control

- Assumes conflicts are rare
- Advantages:
 - No locks used => no lock overhead
 - Deadlock free
 - Possibly maintain high concurrency
 - Reads unrestricted

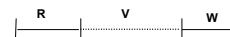
10/07/09

EECS 584, Fall 2009

5

Phases of Optimistic Methods

- Read Phase
- Validation Phase
- Write Phase



10/07/09

EECS 584, Fall 2009

6

Read Phase

- No global writes
 - Copy object on first write
 - Subsequent Writes directed to the copy
- Read/Write/Create/Delete sets populated
 - Read set(T_k): Set of objects read by transaction T_k
 - Write set(T_k): Set of objects modified by T_k

10/07/09 EECS 584, Fall 2009 7

Write Phase

How does validation occur ?

- On validation success
 - Local copies made global
 - Created values become accessible
 - Deleted values become inaccessible
- On validation failure
 - Transaction is backed up, restarted as new transaction

10/07/09 EECS 584, Fall 2009 8

Serial Equivalence Criterion

- Criterion to verify correctness of concurrent execution of transactions
- Says:
 - Concurrent execution of transactions is correct if there exists permutation π such that
$$d_{final} = T_{\pi(n)} \circ T_{\pi(n-1)} \circ \dots \circ T_{\pi(2)} \circ T_{\pi(1)}(d_{initial})$$
 - » T_i : $D \rightarrow D$
 - » T_i : transactions
 - » D = set of all possible shared data structures (d)

10/07/09 EECS 584, Fall 2009 9

Validation of Serial Equivalence

- Assign transaction numbers to each transaction
- For each transaction (T_j) with transaction number $t(j)$, and for all T_i with $t(i) < t(j)$, one of 3 conditions must hold
 - $WP(T_i)$ ends before $RP(T_j)$ start
 - $WP(T_i)$ ends before $WP(T_j)$ start AND $WS(T_i)$ doesn't intersect $RS(T_j)$
 - $WP(T_i)$ ends before $RP(T_j)$ end AND $WS(T_i)$ doesn't intersect $RS/WS(T_j)$

10/07/09 EECS 584, Fall 2009 10

Assigning Transaction Numbers

- Each transaction is assigned a unique integer transaction number $t(i)$
- A global counter is maintained
- Usually assigned at the end of read-phase (optimistic)

10/07/09 EECS 584, Fall 2009 11

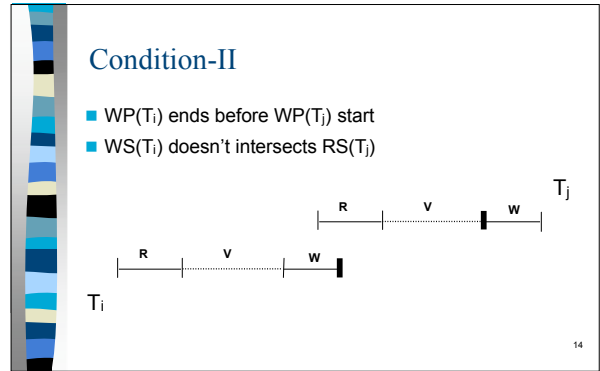
Condition-I

- $WP(T_i)$ end < $RP(T_j)$ start

10/07/09 EECS 584, Fall 2009 12

Bank Example for Condition-I

T1 (\$100 from A to B)	T2 (Add 6% to A, B)
Read Phase Begins	
R(A)	
W(A)	
R(B)	
W(B)	
Read Phase Ends	
Validation Phase Begin	
Validation Phase Ends, write phase ends (Commit changes)	
	Read Phase Begins
	R(A)
	W(A)
	R(B)
	W(B)
	Read Phase Ends
	Validation Phase Begin
	Validation, then Write Phase Ends



Condition-II

T2 ABORTED AND RESTARTED !!!!!

T1 (\$100 from A to B)	T2 (Add 6% to A, B)
Read Phase Begins	
R(A)	
W(A)	
R(B)	
W(B)	
Read Phase Ends	
Validation/Write Phase Begins	
Write phase ends (Commit changes)	
	Read Phase Begins
	R(A) - should read updated value from T1
	W(A)
	R(B)
	W(B)
	Read Phase Ends
	Validation/Write Phase Begin
	Write Phase Ends

10/07/09 EECS 584, Fall 2009 15

Condition-II

T1, T2 EXECUTE CONCURRENTLY!!!!

T1 (\$100 from A to B)	T2 (Add 6% to C, D)
Read Phase Begins	
R(A)	
W(A)	
R(B)	
W(B)	
Read Phase Ends	
Validation Phase Begin	
Validation Phase Ends, write phase ends (Commit changes to disk)	
	Read Phase Begins
	R(C)
	W(C)
	R(D)
	W(D)
	Read Phase Ends
	Validation Phase Begin
	Validation, then Write Phase Ends

10/07/09 EECS 584, Fall 2009 16

Serial Validation (Implementation)

LATCHES, NOT LOCKS. LIMITS CONCURRENCY

```

begin := (
  create set := empty;
  read set := empty;
  write set := empty;
  delete set := empty;
  start tn := tnc;

  tend :=
  finish tn := tnc;
  valid := true;
  for t from start tn + 1 to finish tn do
    if (write set of transaction with transaction number t intersects read set)
      then valid := false;
  if valid
    then ((write phase) tnc := tnc + 1; tn := tnc);
  if valid
    then (cleanup)
    else (backup);

```

10/07/09 EECS 584, Fall 2009 17

Serial Validation (more concurrency)

STILL LIMITS CONCURRENCY

```

tend := (
  mid tn := tnc;
  valid := true;
  for t from start tn + 1 to mid tn do
    if (write set of transaction with transaction number t intersects read set)
      then valid := false;
  (finish tn := tnc;
  for t from mid tn + 1 to finish tn do
    if (write set of transaction with transaction number t intersects read set)
      then valid := false;
  if valid
    then ((write phase); tnc := tnc + 1; tn := tnc);
  if valid
    then (cleanup)
    else (backup);

```

10/07/09 EECS 584, Fall 2009 18

Parallel Validation

```

tend = (
  finish tn := tnc;
  finish active := (make a copy of active);
  active := active ∪ {id of this transaction});
valid := true;
for t from start tn + 1 to finish tn do
  if (write set of transaction with transaction number t intersects read set)
  then valid := false;
for i ∈ finish active do
  if (write set of transaction Ti intersects read set or write set)
  then valid := false;
if valid
then (
  Write phase:
  tnc := tnc + 1;
  tn := tnc;
  active := active - {id of this transaction});
  (cleanup)
else (
  active := active - {id of transaction});
  (backup));

```

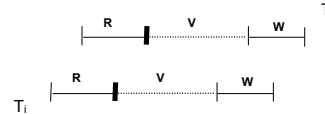
10/07/09

EECS 584, Fall 2009

19

Parallel Validation

- Uses condition III for more concurrency
- $RP(T_i)$ ends before $RP(T_j)$ ends AND $WS(T_i)$ doesn't intersect $RS(T_j)$ or $WS(T_j)$



20

Practical Considerations

- May run out of space on large write-sets
 - Solution
 - Maintain finite number of most recent write-sets
 - If old write-sets are unavailable, validation fails
- Starvation
 - Solution
 - Write Lock the database
 - Run starving transaction to completion

10/07/09

21

Applications

- Query-dominant systems
- Doing concurrent insertions in a B-tree

10/07/09

EECS 584, Fall 2009

22

Analysis of B-tree Insertions

- Expected Size of Read/Write Set small
 - Read/Write Set Objects: Pages of B-tree
 - For order $m = 199$, $N = 200$ million keys, depth < 5
 - For insertions, size of read/write set ≤ 4
- Validation/Write Phase faster than Read Ph.
 - Validation/Write phase can happen in memory, Read phase cannot
- For large B-trees, probability of conflicting insertions: 0.0007

10/07/09

EECS 584, Fall 2009

23

Summary (Optimistic CC Methods)

- Assume conflicts are rare
- Two kinds of concurrency control methods
 - Serial Validation
 - Parallel Validation
- Need to consider starvation/memory constraints
- Better suited for query-dominant systems

10/07/09

24

Thanks !

10/07/09 EECS 584, Fall 2009 25

Example

- Two Transactions in a bank:
 - T1: Transfers \$100 from account A to B
 - T2: Adds 6% interest to both accounts A and B

10/07/09 EECS 584, Fall 2009

Why concurrency?

T1 (\$100 from A to B)	T2 (Add 6% to A, B)
R(A)	
	R(A)
	W(A)
W(A)	
R(B)	
	R(B)
W(B)	
	W(B)

\$100 subtracted from Account A but not added to any account B

- R(A): Read from account A
- W(A): Write to account A

■ Data Integrity and Consistency

10/07/09 EECS 584, Fall 2009

Pessimistic Concurrency Control

T1 (\$100 from A to B)	T2 (Add 6% to A, B)
Lock	
R(A)	
W(A)	
R(B)	
W(B)	
Unlock	
	Lock
	R(A)
	W(A)
	R(B)
	W(B)
	Unlock

■ Assumes conflict will happen

- R(A): Read from account A
- W(A): Write to account A

10/07/09 EECS 584, Fall 2009

Condition-2 (Motivation)

Intersecting

T1 (\$100 from A to B)	T2 (Add 6% to A, B)
Read Phase Begins	
R(A)	Read Set = {A, B}
W(A)	Read Phase Begins
R(B)	R(A)
W(B)	W(A)
Read Phase Ends	R(B)
Validation Phase Begin	W(B)
Validation Phase Ends, write phase ends (Commit changes to disk)	Read Phase Ends
Write Set = {A, B}	Validation Phase Begin
	Validation, then Write Phase Ends

10/07/09 EECS 584, Fall 2009

Condition-2

Not intersecting

T1 (\$100 from A to B)	T2 (Add 6% to C, D)
Read Phase Begins	
R(A)	Read Set = {C, D}
W(A)	Read Phase Begins
R(B)	R(C)
W(B)	W(C)
Read Phase Ends	R(D)
Validation Phase Begin	W(D)
Validation Phase Ends, write phase ends (Commit changes to disk)	Read Phase Ends
Write Set = {A, B}	Validation Phase Begin
	Validation, then Write Phase Ends

10/07/09 EECS 584, Fall 2009