

CS 347:  
Distributed Databases and  
Transaction Processing  
**Notes08: Data Replication**

Hector Garcia-Molina

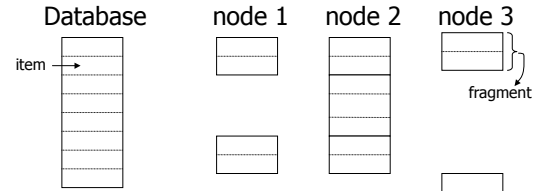
CS 347

Notes08

1

Data Replication

- Reliable net, fail-stop nodes
- The model



CS 347

Notes08

2

- Study one fragment, for time being
- Data replication  $\Rightarrow$  higher availability

CS 347

Notes08

3

Outline

- Basic Algorithms
- Improved (Higher Availability) Algorithms
- Multiple Fragments & Other Issues

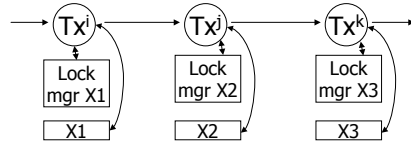
CS 347

Notes08

4

Basic Solution (for C.C.)

- Treat each copy as an independent data item



Object X has copies X1, X2, X3

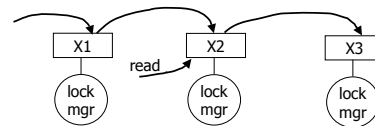
CS 347

Notes08

5

- Read(X):

- get shared X1 lock
- get shared X2 lock
- get shared X3 lock
- read one of X1, X2, X3
- at end of transaction, release X1, X2, X3 locks

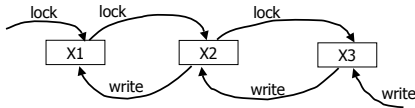


CS 347

Notes08

6

- Write(X):
  - get exclusive X1 lock
  - get exclusive X2 lock
  - get exclusive X3 lock
  - write new value into X1, X2, X3
  - at end of transaction, release X1, X2, X3 locks

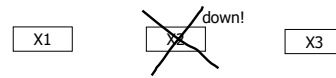


CS 347

Notes08

7

- Correctness OK
  - 2PL  $\Rightarrow$  serializability
  - 2PC  $\Rightarrow$  atomic transactions
- Problem: Low availability



➡ cannot access X!

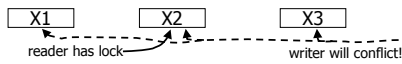
CS 347

Notes08

8

### Basic Solution — Improvement

- Readers lock and access a single copy
- Writers lock all copies and update all copies



- Good availability for reads
- Poor availability for writes

CS 347

Notes08

9

### Reminder

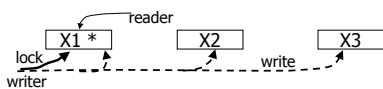
- With basic solution
  - use standard 2PL
  - use standard commit protocols

CS 347

Notes08

10

### Variation on Basic: Primary copy



- Select primary site (static for now)
- Readers lock and access primary copy
- Writers lock primary copy and update all copies

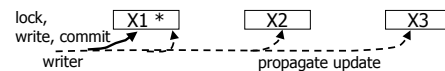
CS 347

Notes08

11

### Commit Options for Primary Site Scheme

- Local Commit



Write(X):

- Get exclusive X1\* lock
- Write new value into X1\*
- Commit at primary; get sequence number
- Perform X2, X3 updates in sequence number order

CS 347

Notes08

12

### Example t = 0

X1: 0 \*  
Y1: 0  
Z1: 0

X2: 0  
Y2: 0  
Z2: 0

T1: X ← 1; Y ← 1;  
T2: Y ← 2;  
T3: Z ← 3;

CS 347

Notes08

13

### Example t = 1

X1: 1 \*  
Y1: 1  
Z1: 3

X2: 0  
Y2: 0  
Z2: 0

T1: X ← 1; Y ← 1; ← active at node 1  
T2: Y ← 2; ← waiting for lock at node 1  
T3: Z ← 3; ← active at node 1

CS 347

Notes08

14

### Example t = 2

X1: 1 \*  
Y1: 2  
Z1: 3

X2: 0  
Y2: 0  
Z2: 0

#2: X ← 1; Y ← 1

T1: X ← 1; Y ← 1; ← committed  
T2: Y ← 2; ← active at 1  
T3: Z ← 3; ← committed

#1: Z ← 3

CS 347

Notes08

15

### Example t = 3

X1: 1 \*  
Y1: 2  
Z1: 3

X2: 1  
Y2: 2  
Z2: 3

#1: Z ← 3

#2: X ← 1; Y ← 1

#3: Y ← 2

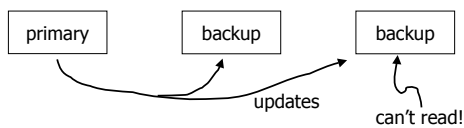
T1: X ← 1; Y ← 1; ← committed  
T2: Y ← 2; ← committed  
T3: Z ← 3; ← committed

CS 347

Notes08

16

### What good is RPWP-LC?



Answer: Can read "out-of-date" backup copy  
(also useful with 1-safe backups... later)

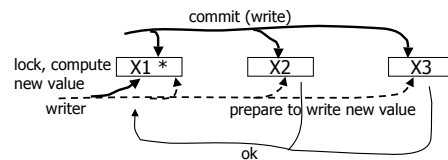
CS 347

Notes08

17

### Commit Options for Primary Site Scheme

#### • Distributed Commit



CS 347

Notes08

18

### Example

X1: 0  
Y1: 0  
Z1: 0

X2: 0  
Y2: 0  
Z2: 0

T1:  $X \leftarrow 1$ ;  $Y \leftarrow 1$ ;  
T2:  $Y \leftarrow 2$ ;  
T3:  $Z \leftarrow 3$ ;

CS 347

Notes08

19

### Basic Solution

- Read lock all; write lock all: RAWA
- Read lock one; write lock all: ROWA
- Read and write lock primary: RPWP
  - local commit: LC
  - distributed commit: DC

CS 347

Notes08

20

### Comparison

N = number of nodes with copies  
P = probability that a node is operational

	Probability can read	Probability can write
RAWA		
ROWA		
RPWP:LC		
RPWP:DC		

CS 347

Notes08

21

### Comparison

N = number of nodes with copies  
P = probability that a node is operational

	Probability can read	Probability can write
RAWA	$p^N$	$p^N$
ROWA	$1 - (1-p)^N$	$p^N$
RPWP:LC	$p$	$p$
RPWP:DC	$p$	$p^N$

CS 347

Notes08

22

### Comparison

N = 5 = number of nodes with copies  
P = 0.99 = probability that a node is operational

	Read Prob.	Write Prob.
RAWA	0.9510	0.9510
ROWA	1.0000	0.9510
RPWP:LC	0.9900	0.9900
RPWP:DC	0.9900	0.9510

CS 347

Notes08

23

### Comparison

N = 100 = number of nodes with copies  
P = 0.99 = probability that a node is operational

	Read Prob.	Write Prob.
RAWA	0.3660	0.3660
ROWA	1.0000	0.3660
RPWP:LC	0.9900	0.9900
RPWP:DC	0.9900	0.3660

CS 347

Notes08

24

## Comparison

N = 5 = number of nodes with copies  
 P = 0.90 = probability that a node is operational

	Read Prob.	Write Prob.
RAWA	0.5905	0.5905
ROWA	1.0000	0.5905
RPWP:LC	0.9000	0.9000
RPWP:DC	0.9000	0.5905

CS 347

Notes08

25

## Outline

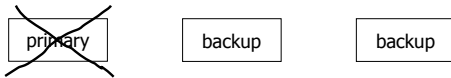
- Basic Algorithms
- Improved (Higher Availability) Algorithms
  - ➔ – Mobile Primary
  - Available Copies
- Multiple Fragments & Other Issues

CS 347

Notes08

26

## Mobile Primary (with RPWP)



- (1) Elect new primary
- (2) Ensure new primary has seen all previously committed transactions
- (3) Resolve pending transactions
- (4) Resume processing

CS 347

Notes08

27

## (1) Elections

- Can be tricky...
- One idea:
  - Nodes have IDs
  - Largest ID wins

CS 347

Notes08

28

## (1) Elections: One scheme

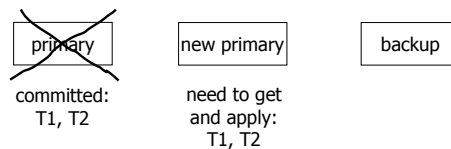
- Broadcast "I want to be primary, ID=X"
- Wait long enough so anyone with larger ID can stop my takeover
- If I see "I want to be primary" message with smaller ID, kill that takeover
- After wait without seeing bigger ID, I am new primary!

CS 347

Notes08

29

## (2) Ensure new primary has previously committed transactions



⇒ cannot use local commit (RPWP:LC)

CS 347

Notes08

30

### (3) Resolve pending transactions

~~primary~~      new primary      backup  
 T3?                      T3 in "W" state      T3 in "W" state

⇒ should not use blocking commit

CS 347                      Notes08                      31

### Failed Nodes: Example

now:      primary P1      backup P2      backup P3  
                  ↓                      -down-                      -down-  
                  commits T1

---

later:      P1      P2      P3  
                  -down-      -down-      -down-

---

later still:      P1      primary P2      backup P3  
                  -down-                      ↓                      -down-  
                                                       commit T2 (unaware of T1!)

CS 347                      Notes08                      32

### 3PC takes care of this problem!

- Option A
  - Failed node waits for:
    - commit info from active node, or
    - all nodes are up and recovering
- Option B
  - Majority voting

CS 347                      Notes08                      33

### Node Recovery

- All transactions have commit sequence number
- Active nodes save update values "as long as necessary"
- Recovering node asks active primary for missed updates; applies in order

CS 347                      Notes08                      34

### Example: Majority Commit

state:	C1	C2	C3
C	T1,T2,T3	T1,T2	T1,T3
P		T3	T2
W	T4	T4	T4

CS 347                      Notes08                      35

### Example: Majority Commit

- t1: C1 fails
- t2: C2 new primary
- t3: C2 commits T1, T2, T3; aborts T4
- t4: C2 resumes processing
- t5: C2 commits T5, T6
- t6: C1 recovers; asks C2 for latest state
- t7: C2 sends committed and pending transactions; C2 involves C1 in any future transactions

CS 347                      Notes08                      36

### 2-safe vs. 1-safe Backups

- Up to now we have covered 2-safe backups (RPWP:DC):

(1) T end work  
 (2) send data  
 (3) get acks  
 (4) commit

CS 347      Notes08      37

### Guarantee

- After transaction T commits at primary, any future primary will "see" T

now:  
 primary backup 1 backup 2  
 T1, T2, T3

later:  
~~primary~~ next primary backup 2  
 T1, T2, T3, T4

CS 347      Notes08      38

### Performance Hit

- 3PC is very expensive
  - many messages
  - locks held longer (less concurrency) [Note: group commit may help]
- Can use 2PC
  - may have blocking
  - 2PC still expensive [up to 1 second reported]

CS 347      Notes08      39

### Alternative: 1-safe (RPWP:LC)

- Commit transactions unilaterally at primary
- Send updates to backups as soon as possible

(1) T end work  
 (2) T commit  
 (3) send data  
 (4) purge data

CS 347      Notes08      40

### Problem: Lost Transactions

now:  
 primary backup 1 backup 2  
 T1, T2, T3 T1 T1

later:  
~~primary~~ next primary backup 2  
 T1, T4, T5 T1, T4

CS 347      Notes08      41

### Claim

- Lost transaction problem tolerable
  - failures rare
  - only a "few" transactions lost

CS 347      Notes08      42

### Primary Recovery with 1-safe

- When failed primary recovers, need to "compensate" for missed transactions

now:

<del>primary</del> T1, T2, T3	next primary T1, T4, T5	backup 2 T1, T4
----------------------------------	----------------------------	--------------------

---

later:

backup 3 T1, T2, T3, T3 <sup>-1</sup> , T2 <sup>-1</sup> , T4, T5 compensation	next primary T1, T4, T5	backup 2 T1, T4, T5
--	----------------------------	------------------------

CS 347                      Notes08                      43

### Log Shipping

- "Log shipping:" propagate updates to backup

```

    graph LR
      primary[primary] -- log --> backup[backup]
  
```

- Backup replays log
- How to replay log efficiently?
  - e.g., elevator disk sweeps
  - e.g., avoid overwrites

CS 347                      Notes08                      44

### So Far in Data Replication

- RAWA
- ROWA
- Primary copy
  - static
    - local commit
    - distributed commit
  - mobile primary
    - 2-safe (distributed commit) blocking or non-blocking
    - 1-safe (local commit)

CS 347                      Notes08                      45

### Outline

- Basic Algorithms
- Improved (Higher Availability) Algorithms
  - Mobile Primary
  - ➔ Available Copies
- Multiple Fragments & Other Issues

CS 347                      Notes08                      46

### PC-lock available copies

\* primary  
 X1      X2      X3      X4  
 down

- Transactions write lock at all available copies
- Transactions read lock at any available copy
- Primary site (static) manages U – set of available copies

CS 347                      Notes08                      47

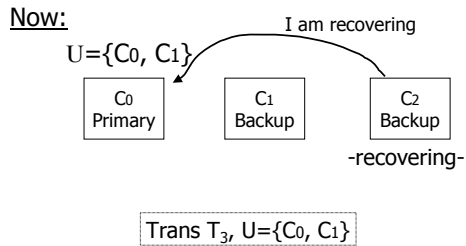
### Update Transaction

- Get U from primary
- Get write locks from U nodes
- Commit at U nodes

$U = \{C_0, C_1\}$   
 C0 Primary      C1 Backup      ~~C2 Backup~~  
 U → Trans T<sub>3</sub>, U = {C<sub>0</sub>, C<sub>1</sub>}

CS 347                      Notes08                      48

### A potential problem - example

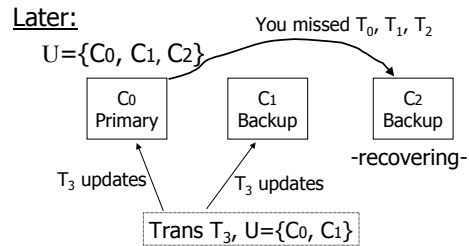


CS 347

Notes08

49

### A potential problem - example



CS 347

Notes08

50

### Solution:

- Initially transaction T gets copy of U' of U from primary (or uses cached value)
- At commit of T, check U' with current U at primary (if different, abort T)

CS 347

Notes08

51

### Solution Continued

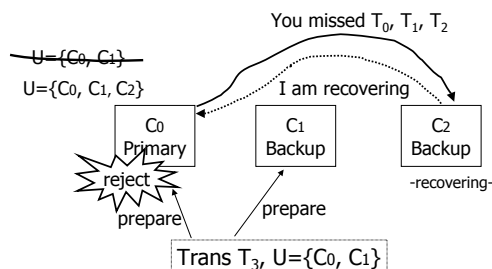
- When C<sub>x</sub> recovers:
  - request missed and pending transactions from primary (primary updates U)
  - set write locks for pending transactions
- Primary polls nodes to detect failures (updates U)

CS 347

Notes08

52

### Example Revisited



CS 347

Notes08

53

### Available Copies — No Primary

- Let all nodes have a copy of U (not just primary)
- To modify U, run a special atomic transaction at all available sites (use commit protocol)
  - E.g.: U<sub>1</sub>={C<sub>1</sub>, C<sub>2</sub>} → U<sub>2</sub>={C<sub>1</sub>, C<sub>2</sub>, C<sub>3</sub>} only C<sub>1</sub>, C<sub>2</sub> participate in this transaction
  - E.g.: U<sub>2</sub>={C<sub>1</sub>, C<sub>2</sub>, C<sub>3</sub>} → U<sub>3</sub>={C<sub>1</sub>, C<sub>2</sub>} only C<sub>1</sub>, C<sub>2</sub> participate in this transaction

CS 347

Notes08

54

- Details are tricky...
- What if commit of U-change blocks?

CS 347

Notes08

55

### Node Recovery (no primary)

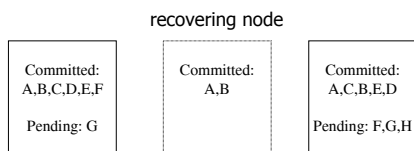
- Get missed updates from any active node
- No unique sequence of transactions
- If all nodes fail, wait for
  - all to recover
  - majority to recover

CS 347

Notes08

56

### Example



- How much information (update values) must be remembered? By whom?

CS 347

Notes08

57

### Correctness with replicated data



S1:  $r_1[X_1] \rightarrow r_2[X_2] \rightarrow w_1[X_1] \rightarrow w_2[X_2]$   
 $\Rightarrow$  Is this schedule serializable?

CS 347

Notes08

58

### One copy serializable (1SR)

A schedule S on replicated data is 1SR if it is equivalent to a serial history of the same transactions on a one-copy database

CS 347

Notes08

59

### To check 1SR

- Take schedule
- Treat  $r_i[X_j]$  as  $r_i[X]$       $X_j$  is copy of X  
     $w_i[X_j]$  as  $w_i[X]$
- Compute P(S)
- If P(S) acyclic, S is 1SR

CS 347

Notes08

60

Example

S1:  $r_1[X_1] \rightarrow r_2[X_2] \rightarrow w_1[X_1] \rightarrow w_2[X_2]$   
 S1':  $r_1[X] \rightarrow r_2[X] \rightarrow w_1[X] \rightarrow w_2[X]$

S1 is not 1SR!

CS 347 Notes08 61

Second example

S2:  $r_1[X_1] \rightarrow w_1[X_1] \rightarrow w_1[X_2]$   
 $r_2[X_1] \rightarrow w_2[X_1] \rightarrow w_2[X_2]$   
 S2':  $r_1[X] \rightarrow w_1[X] \rightarrow w_1[X]$   
 $r_2[X] \rightarrow w_2[X] \rightarrow w_2[X]$

P(S2):  $T_1 \rightarrow T_2$

S2 is 1SR

CS 347 Notes08 62

Second example

S2:  $r_1[X_1] \rightarrow w_1[X_1] \rightarrow w_1[X_2]$   
 $r_2[X_1] \rightarrow w_2[X_1] \rightarrow w_2[X_2]$   
 S2':  $r_1[X] \rightarrow w_1[X] \rightarrow w_1[X]$   
 $r_2[X] \rightarrow w_2[X] \rightarrow w_2[X]$

- Equivalent serial schedule

Ss:  $r_1[X] \rightarrow w_1[X]$   
 $r_2[X] \rightarrow w_2[X]$

CS 347 Notes08 63

Outline

- Basic Algorithms
- Improved (Higher Availability) Algorithms
  - Mobile Primary
  - Available Copies (and 1SR)

➔ Multiple Fragments & Other Issues

CS 347 Notes08 64

Multiple fragments

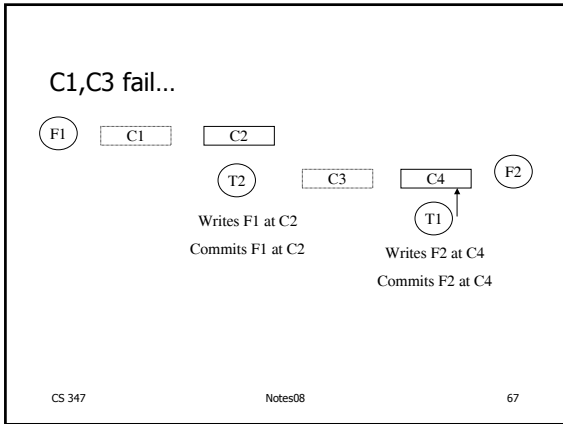
- A transaction spanning multiple fragments must
  - follow locking rules for each fragment
  - commit must involve "majority" in each fragment

CS 347 Notes08 65

- Careful with update transactions that read but do not modify a fragment

Example:

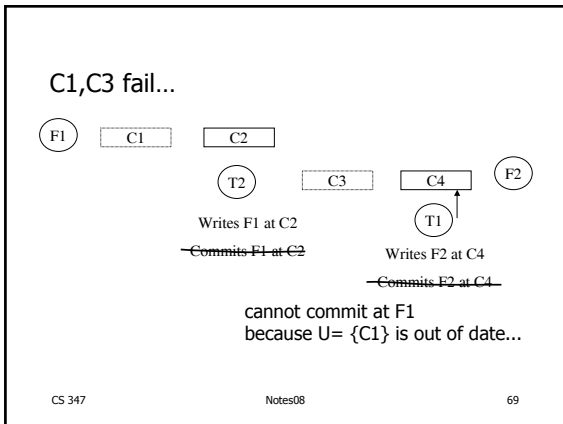
CS 347 Notes08 66



Equivalent history:  
 $r_1[F_1] \ r_2[F_2] \ w_1[F_2] \ w_2[F_1]$   
 not serializable!

Solution: commit at read sites too

CS 347 Notes08 68



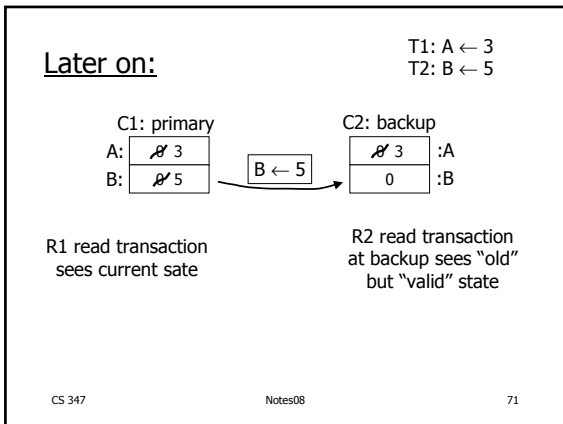
Read-Only Transactions

- Can provide "weaker correctness"
- Does not impact values stored in DB

C1: primary		C2: backup	
A:	0	A:	0
B:	0	B:	0

T1: A ← 3  
T2: B ← 5

CS 347 Notes08 70



States Are Equivalent

- States at Primary:
  - no transactions
  - T1
  - T1, T2
  - T1, T2, T3
- States at Backup:
  - no transactions
  - T1
  - T1, T2
  - T1, T2, T3

at this point in time, backup may be behind...

CS 347 Notes08 72

## Schedule is Serializable

- $S1 = T1 R1 T2 T3 R2 T4 \dots (R1) \dots (R2) \dots$

CS 347

Notes08

73

## Example 2

- A, B have different primaries now
- 1-safe protocol used



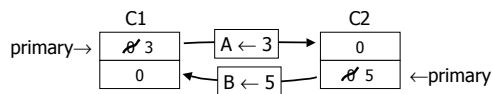
T1: A ← 3  
T2: B ← 5

CS 347

Notes08

74

T1: A ← 3  
T2: B ← 5



### At this time:

- Q1: reads A, B at C1; sees T1 Q1 T2
- Q2: reads A, B at C2; sees T2 Q2 T1

CS 347

Notes08

75

### Eventually:



- Schedule of update transactions is OK:
  - T1 T2 ≡ T2 T1
- Each R.O.T. sees OK schedule:
  - T1 Q1 T2 or T2 Q2 T1
- But there is NO single complete schedule that is "OK"...

CS 347

Notes08

76

- In many cases, such a scenario is OK
- Called weak serializability:
  - update schedule is serializable
  - R.O.T. see committed data

CS 347

Notes08

77

## Data Replication

- RAWA, ROWA
- Primary copy
  - static [local commit or distributed commit]
  - mobile primary [2-safe (2PC or 3PC) or 1-safe]
- Available copies [with or without primary]
- Correctness (1SR)
- Multiple Fragments
- Read-Only Transactions

CS 347

Notes08

78

## Issues

- To centralize control or not?
- How much availability?
- "Weak" reads OK?