Crash recovery

All-or-nothing atomicity & logging
What we’ve learnt so far...

- Consistency in the face of ≥2 copies of data and concurrent accesses
  - Sequential consistency
    - All memory/storage accesses appear executed in a single order by all processes
  - Eventual consistency
    1. All replicas eventually become identical and no writes are lost.
    2. All replicas eventually apply all updates in a single order.

- This class: make data durable across crashes/reboots
Crash at the “wrong time” is problematic

- Examples:
  - Failure during middle of online purchase
  - Failure during “mv /home/jinyang /home/jy”

- What guarantees do applications need?
All-or-nothing atomicity

• All-or-nothing
  – A set of operations either all finish or none at all.
  – No intermediate state exist upon recovery.

• All-or-nothing is one of the guarantees offered by database transactions
Challenges of implementing all-or-nothing

- Crash may occur at any time

- Good normal case performance is desired.
  - Systems usually cache state
An Example

Transfer $1000
From A:$3000
To B:$2000
1st try at all-or-nothing

- Map all file pages in memory
- Modify $A = A - 1000$
- Modify $B = B + 1000$
- Write $A$ to disk
- Write $B$ to disk
2nd try at all-or-nothing

- Read A from $F_{curr}$, read B from $F_{curr}$
- $A = A-1000$; $B = B+1000$
- Write A to $F_{curr}$
- Write B to $F_{curr}$
- Replace $F_{shadow}$ with $F_{curr}$
Problems with the 2nd try

- Multiple transactions might share the same file:
  - Two concurrent transactions:
    - T1: transfer 1000 from A to B
    - T2: transfer 10 from C to D
  - Committing T1 would (falsely) write intermediate state of T2 to disk
3rd try is a charm

- Keep a log of all update actions
- Each action has 3 required operations
SysR: logging

• Merge all transactions into one log
  – Append-only
  – Reduce random access
  – Require linked list of actions within one transaction

• Each log record consists of:
  – Log record length
  – Transaction ID
  – Action ID
  – Timestamp
  – Pointer to previous record in this transaction
  – Action (file name, record name, old & new value)
SysR: logging

- How to commit a transaction?
- SysR logging rules:
  1. Write log record to disk before modifying persistent state
  2. At commit point, append a commit record and force all transaction’s log records to disk
- How to recover from a crash? (no checkpoint)
SysR: checkpoints

• Checkpoints make recovery fast
  – No need to start from a blank state

• How to checkpoint?
  1. Wait till no transactions are in progress (why?)
  2. Write a checkpoint record to log
     • Contains a list of all transactions in progress
  3. Save all files
  4. Atomically save checkpoint by updating root to point to latest checkpoint record (why?)
1. Read most recent checkpoint to learn that T2, T4 are ongoing transactions
2. Read log to learn that T2, T3 are winners and T4 is a loser
3. Read log to undo loser
4. Read log to redo winner
Example using logging

T1
Transfer $1000
From A:$3000
To B:$2000

T2
Transfer $10
From C:$10
To D:$0

sysR

File: F
Rec: A
Old: 3000
New: 2000

File: F
Rec: C
Old: 10
New: 0

Checkpoint
T1,T2

File: F
Rec: B
Old: 2000
New: 3000

Commit
Example recovery

T1
Transfer $1000
From A:$3000
To B:$2000

T2
Transfer $10
From C:$10
To D:$0

sysR

Checkpoint state
A:2000
B:2000
C:0
D:0

File: F
Rec: A
Old: 3000
New: 2000

File: F
Rec: C
Old: 10
New: 0

Checkpt
T1,T2

File: F
Rec: B
Old: 2000
New: 3000

commit
UNDO/REDO logging

• SysR records both UNDO/REDO logs
  – Because a transaction might be very long
    • Must checkpoint w/ ongoing transactions
  – Because a long transaction might be aborted by applications/users
    • Must undo the effects of aborted transactions

• Can we have REDO-only logs for systems w/ “short transactions”?
REDO-only logs

• What’s the logging rule?
  – Append REDO log records before/after flushing state modification?
  – Can uncommitted transactions flush state?

• When can checkpoints be done?
Example using REDO-log

T1

Transfer $1000
From A:$3000
To B:$2000

Checkpoint state
A:3000
B:2000
C:10
D:0

T2

Transfer $10
From C:$10
To D:$0

sysR

Is checkpoint allowed here?

File: F
Rec: A
New: 2000

File: F
Rec: B
New: 3000

File: F
Rec: C
New: 0

commit

Checkpoint state
A:3000
B:2000
C:10
D:0

Recovery goes forward REDO committed actions
REDO-only logs w/o explicit checkpoint

T1
- Transfer $1000
  - From A: $3000
  - To B: $2000

T2
- Transfer $10
  - From C: $10
  - To D: $0

sysR

- File: F
  - Rec: A
    - New: 2000
  - Rec: C
    - New: 0
  - Rec: B
    - New: 3000
- commit

State upon recovery:
- A: 2000
- B: 2000
- C: 10
- D: 0

• Can T1 flush state (A,B)?
• Must T1 flush state (A,B)?
• Can T2 flush state (C)?
• What property must REDO records satisfy?
Case study: disk file systems
FS is a complex data structure

- i-nodes and directory contents are called meta-data
- Also need a free i-node bitmap, a free data block bitmap
Kernel caches used blocks

- Buffer cache holds recently used blocks
- Very effective for reads
  - e.g. access root i-node is extremely fast
- Delay writes
  - Multiple operations can be batched to reduce disk writes
  - Dirty blocks are lost during crash!
Handling crash recovery is hard

- Dangers if crash during meta-data modification
  - Files/dirs disappear completely
  - Files appear when they shouldn’t
  - Files have content belonging to different files
- Dangers of crashing during file content modification
  - Some writes are lost
  - File content are a mix of old and new data
Goal of FS recovery

• Leave file system in a good state w.r.t. meta-data
• It is okay to lose a few operations
  – To tradeoff for better performance during normal operation
A strawman recovery

- The *fsck* program
  1. Descend the FS tree
  2. Remembers allocated i-nodes & blocks
  3. Initialized free i-node & data bitmaps based on step 2.
  4. Also checks for invariants like:
     1. block used by two files
     2. file length != number of blocks etc.
  5. Prompt user if problem cannot be fixed
Example crash problems

User program

```
f = create("d/f", 0666);
write(fd, "hello", 5);
unlink("d/f");
```

File system writes

1. i-node bitmap (Get a free i-node for “f”)
2. “f”’s i-node (write owner etc.)
3. “d”’s dir content (add “f” to i-number mapping)
4. “d”’s i-node (update length & mtime)
5. Block bitmap (get a free block for f’s data)
6. Data block
7. “f”’s i-node (add block to list, update mtime & length)
8. “d”’s content (remove “f” entry)
9. “d”’s i-node (update length, mtime)
10. i-node bitmap
11. block bitmap
FS uses write-back cache

• If every write goes to disk, how fast?
  – 10 ms per modification, 70 ms/file --> 14 files/s
• FS only writes to cache
• When cache fills up with dirty blocks, flush some to disk
  – Writes 1,2,3,4,5 and 7 are amortized over many files
Can we recover with a write-back cache?

• Write-back cache may write to disk in any order.

• Worst case scenarios:
  – A few dirty blocks are flushed to disk, then crash, recover.
Example crash problems

- Wrote 1-8
- Wrote just 3
- Wrote 1-7 and 10

```c
fd = create("d/f", 0666);
write(fd, "hello", 5);
unlink("d/f");
```
A more serious crash

```c
unlink("d/f1");
create("d/f2");
```

- Create happens to re-use i-node freed by unlink
- Only second write of “d” content goes to disk
  - #3: update “d”’s content to add “f2” to i-number mapping
- Recovery:
  - Nothing to fix
  - But file “f2” has “f1” content
  - Serious **undetected** inconsistency
FS needs all-or-nothing metadata update

• How Cedar performs FS operations:
  – Update name table B-tree in memory
  – Append name table modification to in-memory (REDO) log

• When is in-memory log forced to disk?
  – Group commit, every 1/2 second
  – Why?
Cedar’s logging

- When can modified disk cache pages be written to disk?
  - Before writing the log records?
  - After?

- What if it runs out of log space?
  - Flush parts of log to disk, re-use flushed log space
Cedar’s log space reclamation

- Before reclaiming oldest 3rd, flush all its records to disk if the page is not found in later 3rds
Cedar’s recovery

• Recovery re-dos log records
• What’s the state of FS after recovery?
  – Are all completed operations before crash in the recovered state?
  – Cedar recovers a prefix of completed operations
Cedar only logs meta-data ops

- Why not log data?
- What might happen if Cedar crashes while modifying file?
Cedar is fast

• Cedar does 1/7 I/Os for small creates than its predecessor