IT308: Operating Systems

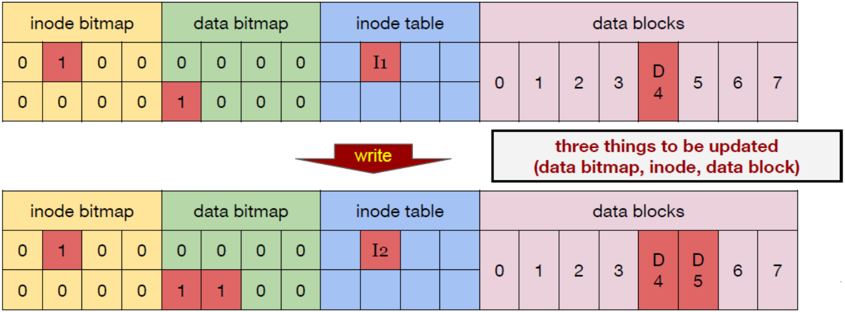
Journaling File Systems

Data persistency

- Data stay after you power-off the computer
  - that's just a property of disks
- If the computer crashes during the file system read/write, we desire that data on disk are still consistent
  - this is a more interesting problem for OS
  - i.e., the crash-consistency problem

Example: A crash scenario

- a 4KB file on the disk, within a single data block
- then we write an additional 4KB to that file, i.e., adding a data block to the file
- if everything went well . . .



If the computer crashed during the write

- What are all the possible inconsistent states that the FS can be in after the crash?
- Only one thing updated

If the computer crashed during the write

- What are all the possible inconsistent states that the FS can be in after the crash?
- Only one thing updated
  - Case 1: Just the data block is updated, but not the data bitmap and inode
  - Case 2: Just the inode is updated, but but not the data bitmap and data block
  - Case 3: Just the data bitmap is updated, but not the inode and data block

Case 1: only data block is updated

- the data is on disk
- but nobody ever knows, because inode and data bitmap are not updated

Case 1: only data block is updated

- the data is on disk
- but nobody ever knows, because inode and data bitmap are not updated
- the file system itself is still consistent, it is just like nothing happened

Case 1: only data block is updated

- the data is on disk
- but nobody ever knows, because inode and data bitmap are not updated
- the file system itself is still consistent, it is just like nothing happened
- No need to fix anything

Case 2: only inode is updated

- inode has data block pointer pointing to an unwritten data block
- if we trust the inode, we will read garbage data

Case 2: only inode is updated

- inode has data block pointer pointing to an unwritten data block
- if we trust the inode, we will read garbage data
- also there is inconsistency between data bitmap and the inode
    - inode says that the data block 5 is used, but data bitmap say it is not

Case 2: only inode is updated

- inode has data block pointer pointing to an unwritten data block
- if we trust the inode, we will read garbage data
- also there is inconsistency between data bitmap and the inode
  - inode says that the data block 5 is used, but data bitmap say it is not
  - if not fixed, could allocate block 5 again and overwrite its data by mistake

Case 3: only data bitmap is updated

- inode is not pointing data block 5, so no risk of reading garbage data
- data bitmap says data block 5 is used, but in fact it is not

Case 3: only data bitmap is updated

- inode is not pointing data block 5, so no risk of reading garbage data
- data bitmap says data block 5 is used, but in fact it is not
- data block 5 will never be used again
- this is called a space leak

If the computer crashed during the write

- What are all the possible inconsistent states that the FS can be in after the crash?
- Only two things updated

If the computer crashed during the write

- What are all the possible inconsistent states that the FS can be in after the crash?
- Only two things updated
  - Case 4: inode and data bitmap updated, but not data block
  - Case 5: inode and data block updated, but not data bitmap
  - Case 6: data bitmap and data block updated, but not onde

Case 4: only inode and data bitmap are updated

- will read garbage data from block 5 again
- but the file system doesn't even realize anything wrong, because the inode and the data bitmap are consistent with each other.

Case 5: only inode and data block are updated

- will NOT read garbage data
- but again, data bitmap and inode are inconsistent with each other
  - inode says that the data block 5 is used, but data bitmap say it is not

Case 5: only inode and data block are updated

- will NOT read garbage data
- but again, data bitmap and inode are inconsistent with each other
  - inode says that the data block 5 is used, but data bitmap say it is not
  - if not fixed, could allocate block 5 again and overwrite its data by mistake

Case 6: only data bitmap and data block are updated

- again, inconsistency between inode and data bitmap
- we know data block 5 is used, but will never know which file uses it

Consistent update problem

- When appending to file, what are the blocks that will be written:

Consistent update problem

- When appending to file, what are the blocks that will be written:
  - data bitmap, file's inode, new data block

Consistent update problem

- When appending to file, what are the blocks that will be written:
  - data bitmap, file's inode, new data block
- If FS is interrupted between writes, inconsistencies may happen
- What can interrupt write operations?
  - Power loss and hard reboot
  - Kernel panic (could be due to bugs not in FS)
  - FS bugs

Consistent update problem

- When appending to file, what are the blocks that will be written:
  - data bitmap, file's inode, new data block
- If FS is interrupted between writes, inconsistencies may happen
- What can interrupt write operations?
  - Power loss and hard reboot
  - Kernel panic (could be due to bugs not in FS)
  - FS bugs
- Need a mechanism to recover from (or fix) inconsistent state

Journaling File Systems

- Basic strategy
  - Before writing the updates (data and metadata), log your intentions

Journaling File Systems

- Basic strategy
  - Before writing the updates (data and metadata), log your intentions
  - Upon a crash, check the log to see what you meant to do

Journaling File Systems

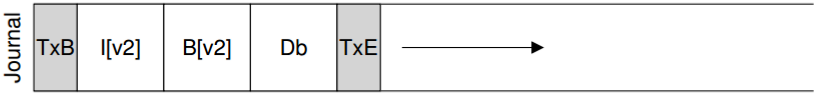
- Basic strategy
  - Before writing the updates (data and metadata), log your intentions
  - Upon a crash, check the log to see what you meant to do
- Known as write-ahead logging in database systems

Data Journaling

- Before writing inode (I[v2]), data bitmap (B[v2]), and data block (Db) to their final destinations, write to the journal

Data Journaling

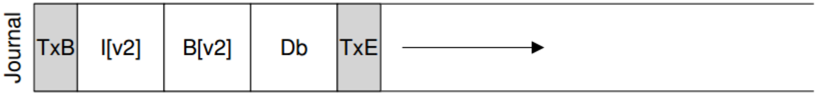
- Before writing inode (I[v2]), data bitmap (B[v2]), and data block (Db) to their final destinations, write to the journal



  - TxB (transaction begin): describes the update, e.g., the final addresses for the blocks, transaction ID

Data Journaling

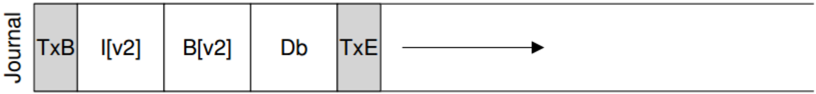
- Before writing inode (I[v2]), data bitmap (B[v2]), and data block (Db) to their final destinations, write to the journal



  - TxB (transaction begin): describes the update, e.g., the final addresses for the blocks, transaction ID
  - Middle three blocks

Data Journaling

- Before writing inode (I[v2]), data bitmap (B[v2]), and data block (Db) to their final destinations, write to the journal



  - TxB (transaction begin): describes the update, e.g., the final addresses for the blocks, transaction ID
  - Middle three blocks
  - TxE (transaction end): mark the end

Sequence of Operations (v1)

1. Journal write
   - Write the transaction, including a transaction-begin block, all pending data and metadata updates, and a transaction-end block, to the log;
   - Wait for these writes to complete.

Sequence of Operations (v1)

1. Journal write
   - Write the transaction, including a transaction-begin block, all pending data and metadata updates, and a transaction-end block, to the log;
   - Wait for these writes to complete.
2. Checkpoint
   - Write the pending metadata and data updates to their final locations in the file system.

How to write the journal?

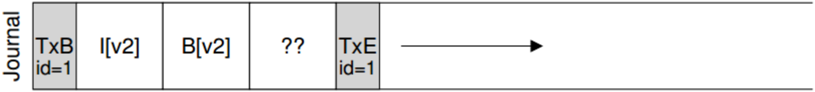
- Write set of blocks: e.g., TxB, I[v2], B[v2], Db, TxE
- Issue one block by one block
    - too slow!

How to write the journal?

- Write set of blocks: e.g., TxB, I[v2], B[v2], Db, TxE
- Issue one block by one block
  - too slow!
- Issue five blocks at one

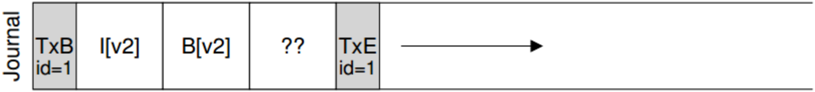
How to write the journal?

- Write set of blocks: e.g., TxB, I[v2], B[v2], Db, TxE
- Issue one block by one block
  - too slow!
- Issue five blocks at one
  - Disk can write blocks out of order
  - Example: Data block not written



How to write the journal?

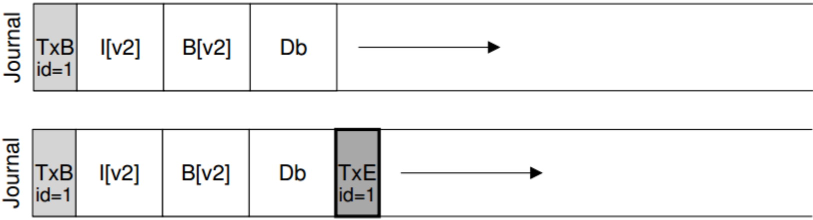
- Write set of blocks: e.g., TxB, I[v2], B[v2], Db, TxE
- Issue one block by one block
  - too slow!
- Issue five blocks at one
  - Disk can write blocks out of order
  - Example: Data block not written



  - Why is this a problem?

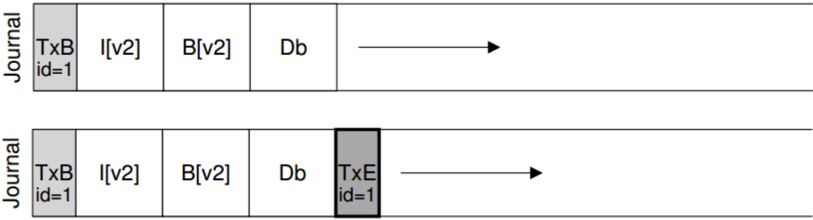
How to write the journal?

- Write in two steps
  - Write everything but TxE first; wait for those write to land
  - Then write TxEnd atomically (make it a single 512-byte block)



How to write the journal?

- Write in two steps
  - Write everything but TxE first; wait for those write to land
  - Then write TxEnd atomically (make it a single 512-byte block)



- Guaranteed to have a valid journal entry

Sequence of Operations (v2)

1. Journal write
   - Write the contents of the transaction (including TxB, metadata, and data) to the log;
   - Wait for these writes to complete.
2. Journal commit
   - Write the transaction commit block (containing TxE) to the log;
   - Wait for write to complete; transaction is said to be committed.
3. Checkpoint
   - Write the contents of the update (metadata and data) to their final on-disk locations.

Recovery

- A crash can happen at any time
- If crash occurs before journal commit
  - do nothing (as if operation never occured)

Recovery

- A crash can happen at any time
- If crash occurs before journal commit
  - do nothing (as if operation never occured)
- If crash occurs after journal commit
  - can replay the log to fix missing writes

Recovery

- A crash can happen at any time
- If crash occurs before journal commit
    - do nothing (as if operation never occured)
- If crash occurs after journal commit
    - can replay the log to fix missing writes
    - worst-case: write some blocks again

Making the Log Finite

- We need to write all data to journal
- What if the log is full?
  - Recovery takes longer to replay everything in the log
  - No further transactions can happen

Making the Log Finite

- We need to write all data to journal
- What if the log is full?
  - Recovery takes longer to replay everything in the log
  - No further transactions can happen
- So, when do you not need journal entries anymore?

Making the Log Finite

- We need to write all data to journal
- What if the log is full?
  - Recovery takes longer to replay everything in the log
  - No further transactions can happen
- So, when do you not need journal entries anymore?
  - After checkpoint the transaction in the journal is not useful anymore, so the space can be freed

Sequence of Operations (v3)

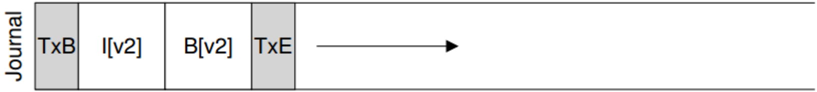
1. Journal write
   - Write the contents of the transaction (including TxB, metadata, and data) to the log;
   - Wait for these writes to complete.
2. Journal commit
   - Write the transaction commit block (containing TxE) to the log;
   - Wait for write to complete; transaction is said to be committed.
3. Checkpoint
   - Write the contents of the update (metadata and data) to their final on-disk locations.
4. Free
   - Some time later, mark the transaction free in the journal.

Metadata Journaling

- Data journaling writes all data twice, once for real
  - Wasteful and slow

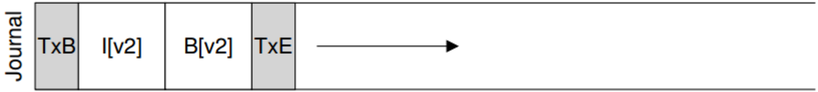
Metadata Journaling

- Data journaling writes all data twice, once for real
  - Wasteful and slow
- Don't write data blocks to journal; just the inode and bitmap



Metadata Journaling

- Data journaling writes all data twice, once for real
  - Wasteful and slow
- Don't write data blocks to journal; just the inode and bitmap



- When to write Db to disk?

Metadata Journaling

- Say we checkpoint metadata, then write data

Metadata Journaling

- Say we checkpoint metadata, then write data
- If write data fails, the inodes will point to garbage data

Metadata Journaling

- Say we checkpoint metadata, then write data
- If write data fails, the inodes will point to garbage data
- How to solve this problem?

Metadata Journaling

- Say we checkpoint metadata, then write data
- If write data fails, the inodes will point to garbage data
- How to solve this problem?
  - Write data before writing metadata journal

Sequence of Operations (v4)

1/2. Write data
1/2. Metadata journal write
3. Metadata journal commit
4. Checkpoint metadata
5. Free

- If write data fails, then no metadata is written at all, like nothing happened
- If data write succeeds, but metadata write fails, still like nothing happened
- If metadata write succeeds, data must be available.

Data Journaling Timeline

Journal: TxB	Journal: Contents (metadata)	Journal: Contents (data)	Journal: TxE	File System: Metadata	File System: Data
issue	**issue**	**issue**			
complete					
	complete				
		complete			
			issue		
			complete		
				issue	**issue**
					complete
				complete	

Metadata Journaling Timeline

Journal: TxB	Journal: Contents (metadata)	Journal: TxE	File System: Metadata	File System: Data
issue	issue			issue
				complete
complete				
	complete			
		issue		
		complete		
			issue	
			complete	