CH14-4-Crash Consistency-FSCK and Journaling

資工三 穆冠蓁 4107056007

* A Detailed Example

1. Crash-Consistency Problem

(1) 問題 : 系統當機會導致硬碟上data structure的不一致(inconsistency)

(2) 解決方法 :

- fsck(file system checker)事後治療

- journaling 事先預防

2. A Detailed Example

(1) 假設已經存在的檔案要新增一個data block，有以下三個資料結構須被寫入: inode(指向new block的 pointer、size)、new data block、data bitmap

(2) 有可能在寫入其中一兩個的時候，系統就當機->造成inconsistency

(3) Case 1 : Only a Single Write Succeeds

- 只有data block 被寫入磁碟(data bitmap 、inode未更新) :

在檔案系統的角度，寫入沒發生過，因此FS is consistent(只有資料遺失)

* 只有inode 被寫入磁碟 : data bitmap 及inode資料不一致(若相信inode，會讀到garbage data) ->inconsistent
* 只有bitmap 被寫入磁碟 : FS is inconsistent(若是相信bitmap，會造成空間浪費，space leak)

(4) Case 2 : Only a Two Write Succeeds

- inode 和 data block 被寫入硬碟 : inconsistent

- bitmap 和 data block 被寫入硬碟 : inconsistent

- inode 和 bitmap 被寫入硬碟 : consistent(但會讀到garbage data)

* Solution #1: The File System Checker: 事後治療

1. 讓檔案系統的不一致發生，並試著修復(Linux fsck)

2. 目標 : 保證file system 的 metadata(Inode+bitmap)是一致的

3. 步驟 :

(1) Superblock : 檢查superblock 的內容是否有異常

(2) Free blocks\_1:data block bitmap

- 掃描inode知道哪些block有被配置，產生一個allocation bitmap

- allocation bitmap 若是與 data block bitmap不一致，fsck選擇信任inode的資訊

(3) Free blocks\_2: inode bitmap

- 檢查所有的inodes，並產生inode bitmap

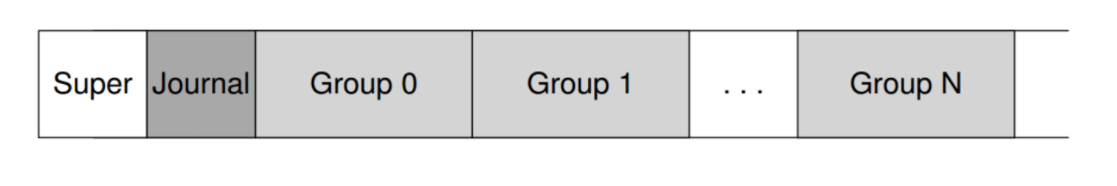
- 將產生的inode bitmap和原本的inode bitmap比較，若是不一致，選擇相信inode的資訊

4. 問題 : 難以實作fsck(須對FS的資料結構很清楚)，而且因為需要掃描整個檔案系統，所以速度很慢

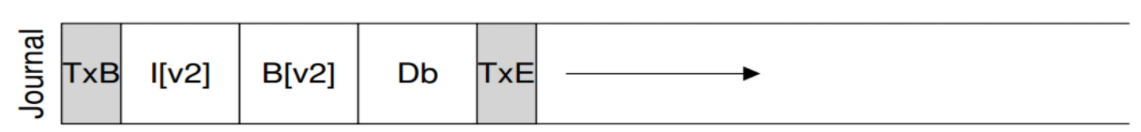
* Solution #2: Journaling (or Write-Ahead Logging):事先預防

1. 又稱write-ahead : 在修改硬碟中的資料結構前，事先寫到log

2. Linux ext3 Journaling File System : 將journal加到原本的檔案系統中



3. Data Journaling



(1) 將inode、bitmap、datablock寫進去前，先寫入log file(journal)

(2) Log structure :

- TxB: Transaction Begin.

- Physical (logical)logging:寫入inode、bitmap、datablock

- TxE: Transaction End

4. Checkpointing

(1) 若是inode、bitmap、datablock都成功寫入log，我們就稱transaction commited

(2) checkpoint :在transaction完成後，更新真實的資料結構

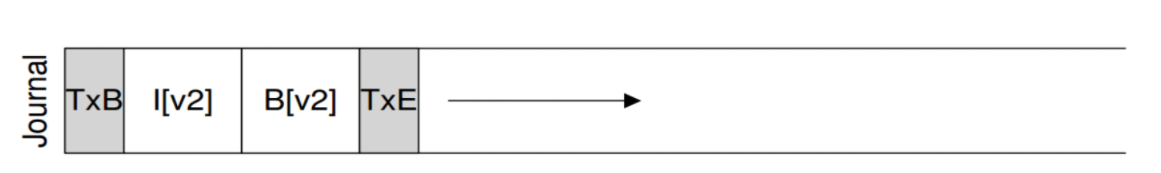
5. Crash Recovery :

(1) Crashe during logging :reboot 之後會清空journal，FS consistent

(2) Crashe during checkpoint : 重新scon log，並重新commit(log 還未清除，表示還未完成checkpoint) FS consistent

6. Recycling Log Space : 當完成checkpoint，FS就會清空Log

7. Metadata Journaling :每次write to disk，我們都要分別將資料寫入journal和FS(因此需寫入兩次)



(1) 問題 : Crash 可能很少發生，但花的成本卻很高(metadata通常比較小，而datablock的寫入因為容量大，通常最花時間)

(2) 解決方法 : Metadata Journaling (只將metadata寫入log)

(3) 將data blocks寫入磁碟的時間點 : Journal write、Data write、Checkpoint

- Journal write ⇨ **Data Write** ⇨ Checkpoint

若是在journal write之後發生crashe(reboot 後會做checkpoint，但datablock仍沒正確寫入)，雖然系統仍是consistent，但會指到garbage data

* Data write ⇨ Journal Write ⇨ Checkpoint

若是data已經寫入datablock 之後發生crashe，reboot後會將未完成的log file清空(系統仍是consistent)，但user寫入的資料卻被忽略