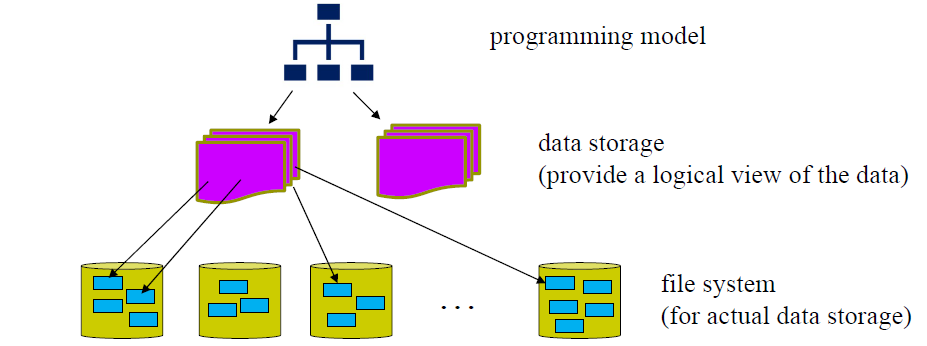
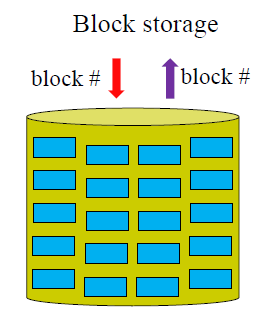
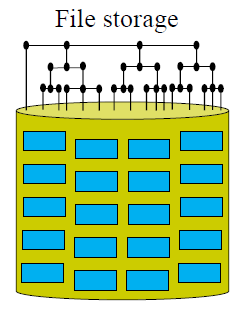
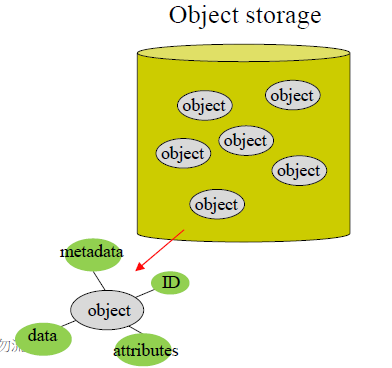
**Large Distributed File Systems**

一個Bigdata Applications需要甚麼?

* programming model
  + 用來processing the data。
    - 平行、分散式處理data
  + Ex: MapReduce
* database-like storage system (logical view)
  + 用來處理資料的 read/write
* File system (存真正的資料)
  + 真正用來存data的地方
  + Ex: GFS (Google File System)
* 整個架構
  + 

三種storage存資料

* Block storage
  + 在實體硬體中一個一個block存資料
  + 
* File storage
  + 用file hierarchy的方式去存
  + 
* Object storage
  + objects containing, in addition to its data, ID, metadata, and attributes (存比較多額外資料)
  + 

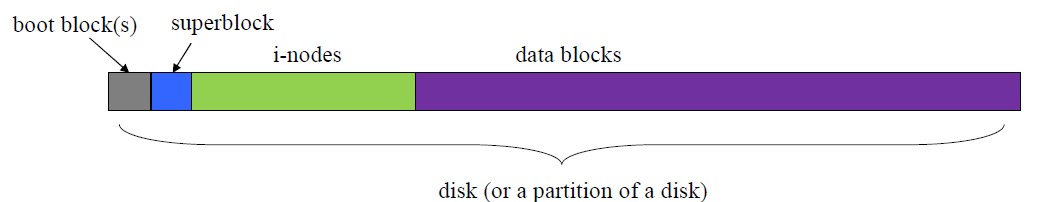
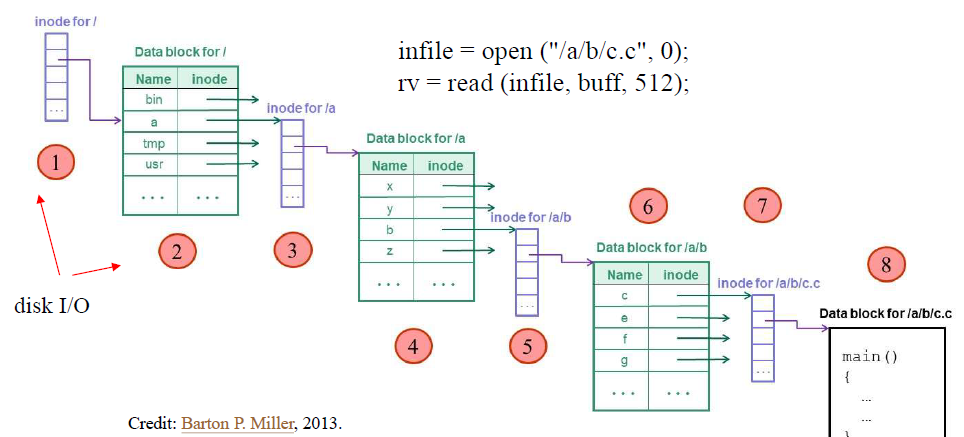
File Systems vs. Database

* File System
  + a collection of raw data files
* Databases
  + are designed particularly for easily organizing, storing and retrieving large volume of data

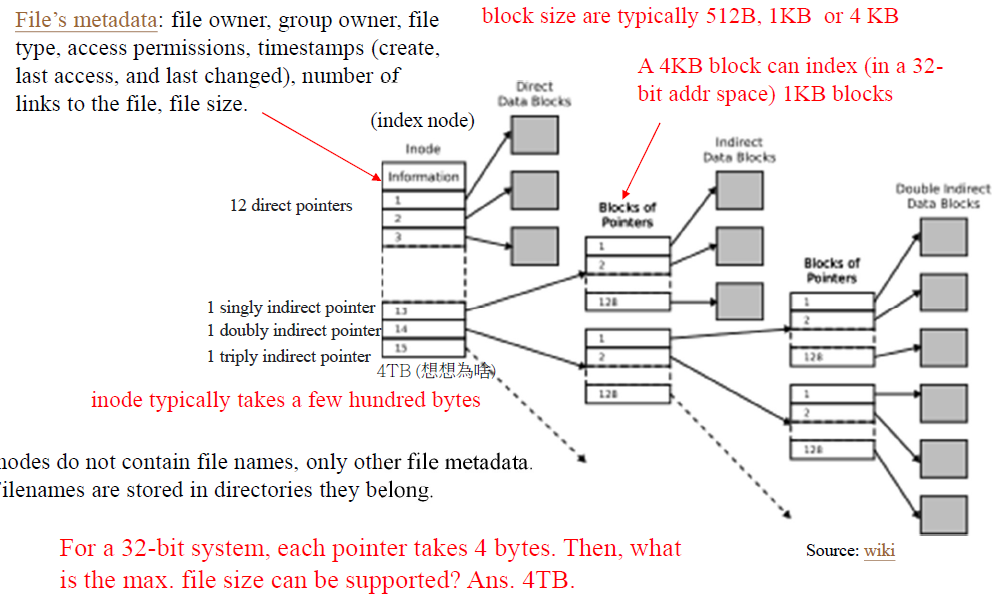
Centralized File Systems 有兩種知名的

* Unix/Linux
* DOS FAT (File Allocation Table)

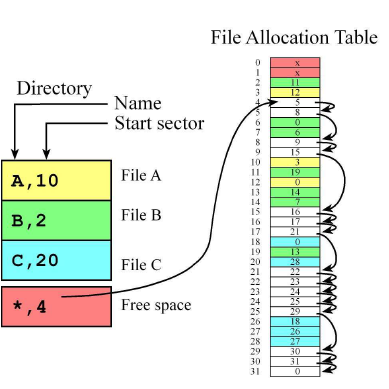
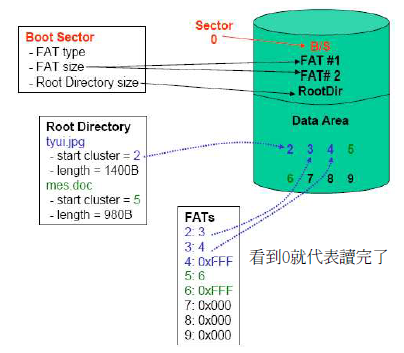
Unix-like File System

* 
* boot block
  + 有boot loader，用來boot OS
* Superblock
  + basic information about the entire file system
* i-nodes
  + holds a pointer to the disk blocks containing the data in the file
  + 就是有pointer指向一個disk block，這個block會包含data的資料
  + 
  + Ex: open一個檔案時，其實就是透過i-node一直去找data block，然後最後找到我們要的檔案
  + Note
    - i-node 0 = null
    - i-node 1 = used to keep track of bad blocks on the disk
    - i-node 2 = root directory

Unix i-node特殊設計

* block會有固定的大小，所以檔案大小會有侷限
  + 通常是512B、1KB、4KB
* 
  + 但用最下面三個indirect pointer可以讓它增加至 4TB

DOS FAT

* 
  + 就是按照順序一直讀，讀到0就代表結束，也是pointer的概念
* 
  + 也有Boot Sector這種開機相關的區塊
    - 裡面存有FAT type、FAT size、Root directory size

這兩個案例都是centralize的系統

* 都有index structure(pointer這種方式?)來找data block
  + 要如何用到分散式系統中?
* 該如何在多台host中維護這種index structure?

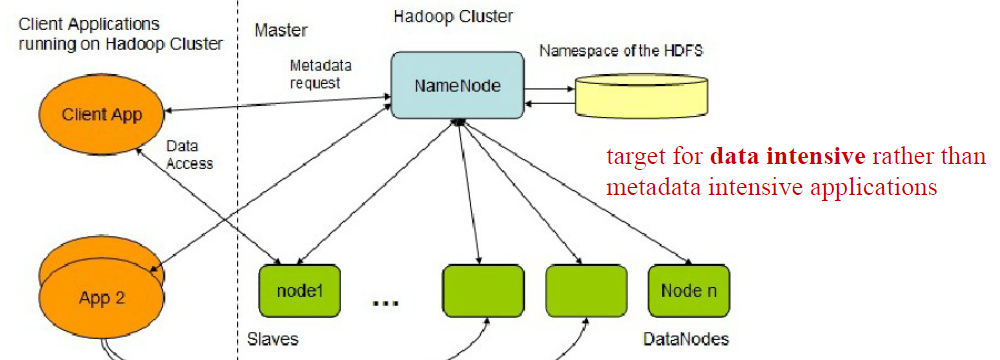
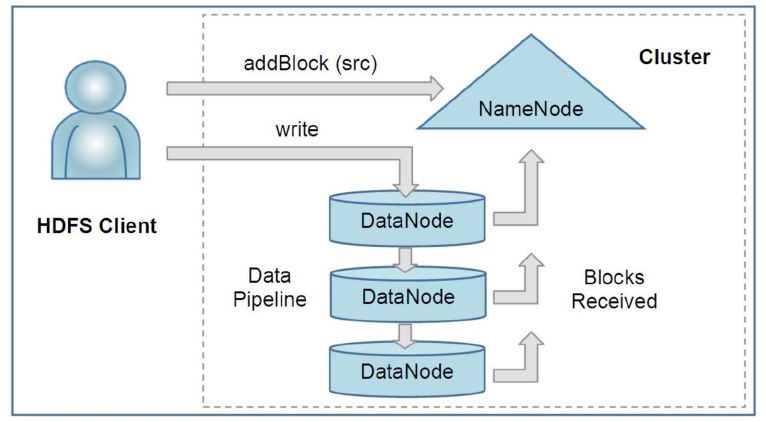
三種大型分散式檔案系統案例

* HDFS - Hadoop Distributed File System (HDFS)
* The Google File System (GFS)
* Ceph Distributed File System

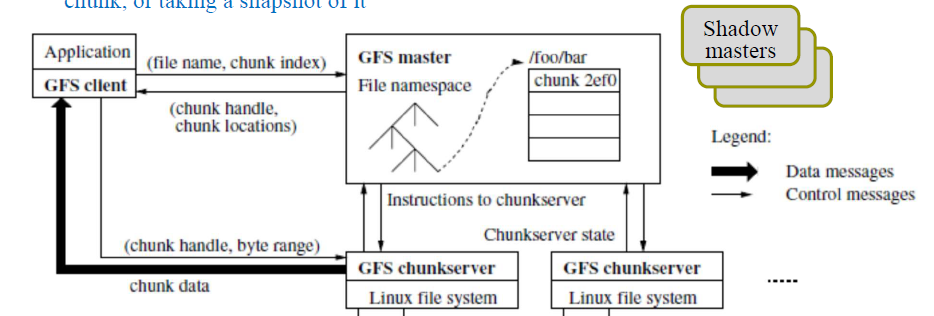
(Apache) Hadoop

* Apache top level project (當初為了進入大數據時代，而開啟的計畫)
* open-source software
  + reliable, scalable, distributed computing and data storage
* a framework that allow for the distributed processing of large data sets (bigdata)
* 是由yahoo發起的, 用 JAVA 寫的

HDFS Design Goals

* highly fault-tolerant, high throughput distributed file system designed to run on commodity hardware (typically run a GNU/Linux OS)
* Not general purpose
  + for batch processing rather than interactive use by users
  + 假設基本上只有write，不會去修改資料
    - 好像是write-once-read-many
    - Ex: mail server，email寄出去就寄出去，不會再去修改了
* 有很多概念是參考GFS
* Large Data Sets: (補) P20
  + 應該是下面會提到的，每個Node的size比較大
  + HDFS就是為了處理大容量的file而產生的設計，而且這樣子對Master Name-node的負擔也比較小
* 有假設: write-once-read-many access model for files (補) P20
  + 這是簡化concurrency write的問題，因為經驗上發現在真實應用場警不太會同時write。(還是可能有問題，因此HDFS沒有符合POSIX
* Master/Slave Architecture
  + 
  + HDFS cluster consists of a single Name-node
    - Name-node底下會有很多Data-Nodes
  + 要找資料就問Name-node，它會去Data-node找資料
    - Data-node就像是之前提到的block?
    - (重點)data block的大小會設計得比較大!
    - 這些block不會像之前只有1KB，而是有1MB之類的大小，這樣Name-node就不用存太多資料
      * 因為每一次都指向一個1MB大小的block
    - 後面有好幾頁PDF可以看更仔細 (補) P22
  + 一些額外常識
    - Slave的node會定期回報Name-node，讓Master知道node還活著 (這個機制稱作Heartbeat)
    - Slave的node通常會有備份 (通常是3份)
    - 如果回報時發現slave node掛了，就會增加備份
    - 系統startup時要做甚麼?
      * Slave data node會告訴Master Name-node自己的狀況
  + Name-node額外的一些工作 (補) P27
    - CheckpointNode
      * 週期性的整合log，讓log不會無限增長?
    - BackupNode
      * 週期性的backup node?
  + 如何選擇備份要放哪?
    - Rack(機架)-aware replica placement policy:
      * Rack就是像Google或AWS那種機房的架子
    - 假設有3個備份要放
      * 1. 放最近的位置 (local)
      * 2. 放在跟第一個rack不同的地方(remote)
      * 3. 有的寫local、有的寫remote
      * 如果還有更多，就隨機放
  + HDFS如何write data?
    - 
      * Client只會write給一個data node，之後他們就自己往下傳遞，data node寫好會回報給Name node

GFS (Google File System)

* Design Goal
  + a scalable, fault-tolerant distributed file system
  + 一樣不是general purpose
    - Google每開發一個新的應用，都會有自己的file system，但底層都是GFS
    - mostly by co-designed applications
* Design Assumptions
  + 跟前面的Hadoop假設一樣(應該說前面跟這裡一樣)
  + Ex: write once, read many
* User interface
  + 主要的操作
    - create, delete, open, close, read, and write
  + Two operation
    - Snapshots
    - Record appends
  + 沒有遵守POSIX (HDFS也沒遵守)，所以會有同步的問題!?
    - POSIX是一種規範
      * Read一定要反映最新的write
      * Write一定是atomic
  + 架構
    - 
    - 跟前面前面基本上一樣
    - GFS master = Name-node
      * 存有每個檔案在哪個chunk上，有多少備份
        + 這就是index structure
    - GFS chunkserver = Data-node
      * 也有heartbeat的機制
      * 多了一些之前沒提到的功能
        + lease management

chunk修改資料時，為了讓每一份replica保持consistent，所以會給其中一個人lease (租約?)，它會變成primary，讓其他replica參照它修改

如果chunk 60秒內沒有回應，就會給其他人

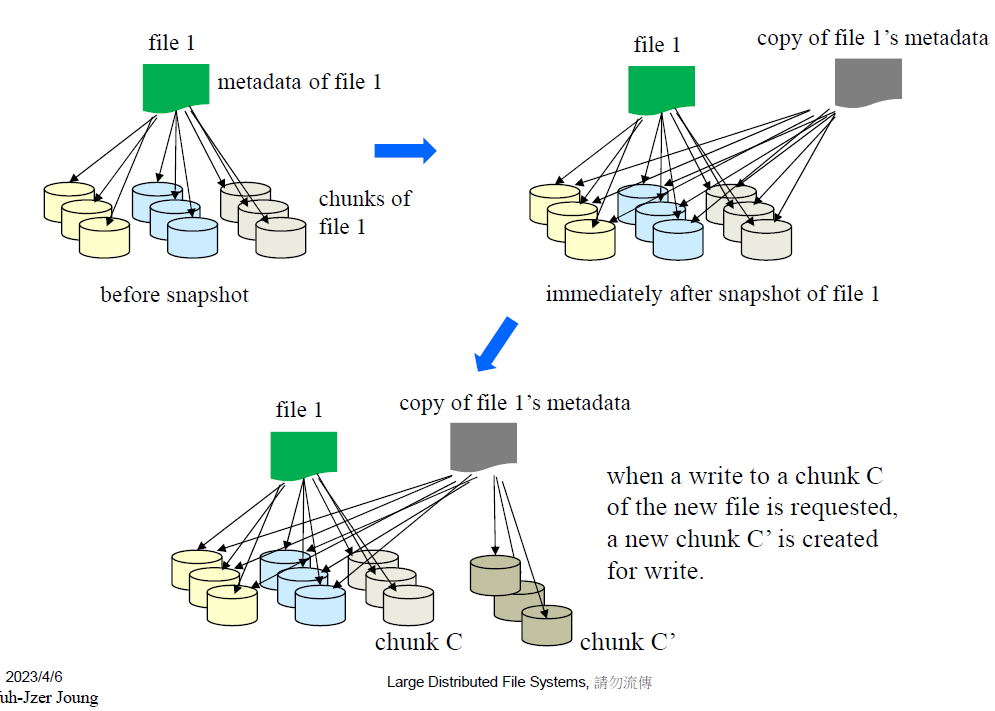
Lease通常有使用期限，時間到了就會過期

這個設計在client端，可以降低master的負擔

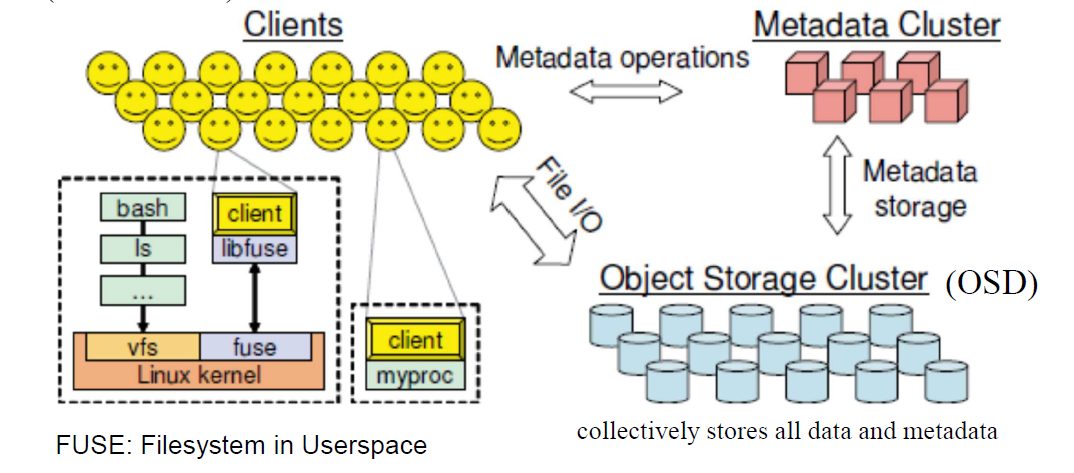
* + - * + garbage collection of orphaned chunks (P62)

因為GFS的檔案都很大，Delete很花時間

因此會直接在Master標註某個chunk要delete，之後有空再慢慢回收資源

* + - * + chunk migration between chunk servers
  + 一些定義
    - Chunk size: 64MB
      * 降低 index 的成本
      * Note: 好像有提到，GFS、HDFS都不是為了處理小檔案而設計的，通常檔案的是GB、TB等級的
  + 備份放哪? (P50)
    - 透過heartbeat告訴master chunk自己在哪
      * 跟HDFS好像不一樣
  + 如何找chunk在哪?
    - 使用B+ Tree的資料結構
  + Snapshot
    - 複製一個檔案或目錄
      * 但通常GFS的檔案大小都很大，該如何處理?
        + Use copy-on-write approach
        + 事實上只有複製metadata (有點不懂，可以再看一次)
        + 
  + P60: locking
* Google Colossus File System
  + 比GFS大100倍的GFS
  + 好像是general purpose了

Ceph & RADOS

* Design Goal:
  + scalable, reliable, and high-performance distributed file system
* 較有符合POSIX
* 比較算是general purpose
* 架構
  + 
  + 為了處理concurrency write (符合POSIX)，因此有多個Master
    - 這裡的Master = Metadata Cluster
* 如何將File放進OSD (Objects Storage Cluster)?
  + P69