Btrfs vs ZFS 實現 snapshot 的差異

目录

Btrfs 和 ZFS 都是開源的寫時拷貝（Copy on Write, CoW）文件系統，都提供了相似的子卷管理和 快照 (snapshot）的功能。網上有不少文章都評價 ZFS 實現 CoW FS 的創新之處，進而想說「 Btrfs 只是 Linux/GPL 陣營對 ZFS 的拙劣抄襲」，或許（在存儲領域人盡皆知而領域外）鮮有人知在 ZFS 之前就有 NetApp 的商業產品 WAFL(Write Anywhere File Layout) 實現了 CoW 語義的文件系統，並且集成了快照和卷管理之類的功能。我一開始也帶着「 Btrfs 和 ZFS 都提供了類似的功能，因此兩者必然有類似的設計」這樣的先入觀念，嘗試去使用這兩個文件系統， 卻經常撞上兩者細節上的差異，導致使用時需要不盡相同的工作流， 或者看似相似的用法有不太一樣的性能表現，又或者一邊有的功能（比如 ZFS 的 inband dedup ， Btrfs 的 reflink ）在另一邊沒有的情況。

爲了更好地理解這些差異，我四處查詢這兩個文件系統的實現細節，於是有了這篇筆記， 記錄一下我查到的種種發現和自己的理解。~~（或許會寫成一個系列？還是先別亂挖坑不填。）~~ 只是自己的筆記，所有參閱的資料文檔都是二手資料，沒有深挖過源碼，還參雜了自己的理解， 於是難免有和事實相違的地方，如有寫錯，還請留言糾正。

Btrfs 的子卷（subvolume）和快照（snapshot）

先從兩個文件系統中（表面上看起來）比較簡單的 btrfs 的子卷（subvolume）和快照（snapshot）說起。 關於子卷和快照的常規用法、推薦佈局之類的話題就不細說了，網上能找到很多不錯的資料，比如 btrfs wiki 的 SysadminGuide 頁 和 Arch wiki 上 Btrfs#Subvolumes 頁都有不錯的參考價值。

在 btrfs 中，存在於存儲媒介中的只有「子卷」的概念，「快照」只是個創建「子卷」的方式， 換句話說在 btrfs 的術語裏，子卷（subvolume）是個名詞，而快照（snapshot）是個動詞。 如果脫離了 btrfs 術語的上下文，或者不精確地隨口說說的時候，也經常有人把 btrfs 的快照命令創建出的子卷叫做一個快照。或者我們可以理解爲， **互相共享一部分元數據（metadata）的子卷互爲彼此的快照（名詞）** ， 那麼按照這個定義的話，在 btrfs 中創建快照（名詞）的方式其實有兩種：

1. 用 `btrfs subvolume snapshot` 命令創建快照
2. 用 `btrfs send` 命令並使用 `-p` 參數發送快照，並在管道另一端接收

`btrfs send` 命令的 `-p` 與 `-c`

這裏也順便提一下 `btrfs send` 命令的 `-p` 參數和 `-c` 參數的差異。 只看 btrfs-send(8) 的描述的話：

> -p <parent>
>
> send an incremental stream from parent to subvol
>
> -c <clone-src>
>
> use this snapshot as a clone source for an incremental send (multiple allowed)

看起來這兩個都可以用來生成兩個快照之間的差分，只不過 -p 只能指定一個「parent」， 而 -c 能指定多個「clone source」。在 unix stackexchange 上有人寫明了這兩個的異同 。使用 -p 的時候，產生的差分首先讓接收端用 subvolume snapshot 命令對 parent 子卷創建一個快照， 然後發送指令將這個快照修改成目標子卷的樣子，而使用 -c 的時候，首先在接收端用 subvolume create 創建一個空的子卷，隨後發送指令在這個子卷中填充內容，其數據塊儘量共享 clone source 已有的數

據。 所以 `btrfs send -p` 在接收端產生是有共享元數據的快照，而 `btrfs send -c` 在接收端產生的是僅僅共享數據而不共享元數據的子卷。

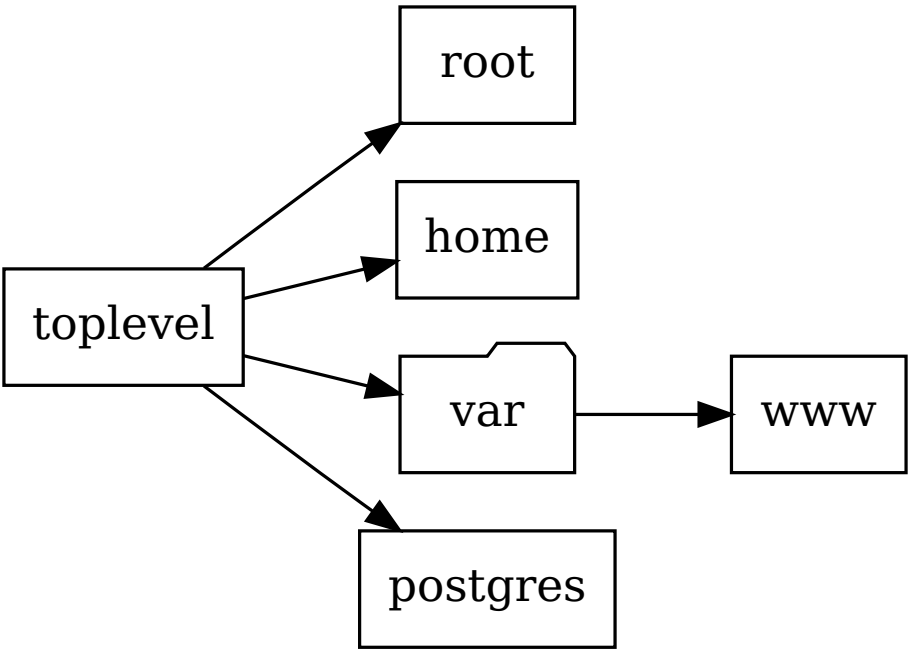
定義中「互相共享一部分 **元數據** 」比較重要，因爲除了快照的方式之外， btrfs 的子卷間也可以通過 reflink 的形式共享數據塊。我們可以對一整個子卷（甚至目錄）執行 `cp -r --reflink=always` ，創建出一個副本，副本的文件內容通過 reflink 共享原本的數據，但不共享元數據，這樣創建出的就不是快照。

說了這麼多，其實關鍵的只是 btrfs 在傳統 Unix 文件系統的「目錄/文件/inode」 這些東西之外只增加了一個「子卷」的新概念，而子卷間可以共享元數據或者數據， 用快照命令創建出的子卷就是共享一部分元數據。於是這個子卷在文件系統中具體是如何記錄的呢？ 舉個例子解釋可能比較好理解：

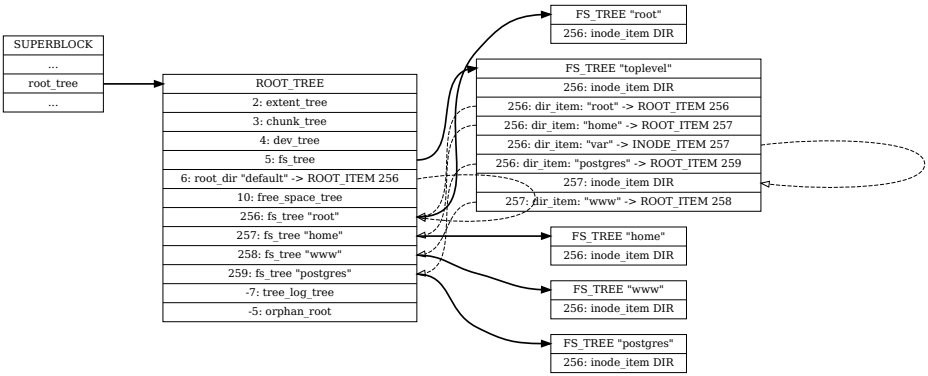
比如在 SysadminGuide 這頁的 Flat 佈局 有個子卷佈局的例子。

```
toplevel         (volume root direc
tory, not to be mounted by default)
    +-- root       (subvolume root
directory, to be mounted at /)
    +-- home       (subvolume root
directory, to be mounted at /home)
    +-- var        (directory)
    |   \-- www    (subvolume root
directory, to be mounted at /var/ww
w)
    \-- postgres   (subvolume root
directory, to be mounted at /var/li
b/postgresql)
```

用圓柱體表示子卷的話畫成圖大概是這個樣子：



首先要說明， btrfs 中大部分長度可變的數據結構都是 CoW B-tree ，一種經過修改適合寫時拷貝的B樹結構，所以在 on-disk format 中提到了很多個樹。這裏的樹不是指文件系統中目錄結構樹，而是 CoW B-tree ，如果不關心B樹細節的話可以把 btrfs 所說的一棵樹理解爲關係數據庫中的一個表， 和數據庫的表一樣 btrfs 的樹的長度可變，然後表項內容根據一個 key 排序。 有這樣的背景之後，上圖例子中的 Flat 佈局在 btrfs 中大概是這樣的數據結構：



上圖中已經隱去了很多和本文無關的具體細節，所有這些細節都可以通過 btrfs inspect-internal 的 dump-super 和 dump-tree 查看到。btrfs 中的每棵樹都可以看作是一個數據庫表，可以包含很多表項，根據 KEY 排序，而 KEY 是 (object_id, item_type, item_offset) 這樣的三元組。每個 object 在樹中用一個或多個 item 描述，同 object_id 的 item 共同描述一個對象（object）。B樹中的 key 不必連續，從而 object_id 也不必連續，只是按大小排序。有一些預留的 object_id 不能用作別的用途， 他們的編號範圍是 -255ULL 到 255ULL，也就是表中前 255 和最後 255 個編號預留。

ROOT_TREE 中包含了到別的所有 tree 的定義，像 2 號 extent_tree ，3 號 chunk_tree ， 4 號 dev_tree ，10 號 free_space_tree ，這些 tree 都是描述文件系統結構非常重要的 tree 。然後在 5 號對象有一個 fs_tree 它描述了整個 btrfs pool 的頂級子卷，也就是圖中叫 toplevel 的那個子卷。除了頂級子卷之外，別的所有子

卷的 object_id 在 256ULL 到 -256ULL 的範圍之間，對子卷而言 ROOT_TREE 中的這些 object_id 也同時是它們的 子卷 id ，在內核掛載文件系統的時候可以用 subvolid 找到它們，別的一些對子卷的操作也可以直接用 subvolid 表示一個子卷。 ROOT_TREE 的 6 號對象描述的不是一棵樹，而是一個名叫 default 的特殊目錄，它指向 btrfs pool 的默認掛載子卷。最初 mkfs 的時候，這個目錄指向 ROOT_ITEM 5 ，也就是那個頂級子卷，之後可以通過命令 `btrfs subvolume set-default` 修改它指向別的子卷，這裏它被改爲指向 ROOT_ITEM 256 亦即那個名叫 "root" 的子卷。

每一個子卷都有一棵自己的 FS_TREE （有的文檔中叫 file tree），一個 FS_TREE 相當於傳統 Unix 文件系統中的一整個 inode table ，只不過它除了包含 inode 信息之外還包含所有文件夾內容。在 FS_TREE 中，object_id 同時也是它所描述對象的 inode 號，所以 btrfs 的子卷有互相獨立的 inode 編號，不同子卷中的文件或目錄可以擁有相同的 inode 。 FS_TREE 中一個目錄用一個 inode_item 和多個 dir_item 描述， inode_item 是目錄自己的 inode ，那些 dir_item 是目錄的內容。 dir_item 可以指向別的 inode_item ，描述普通文件和子目錄， 也可以指向 root_item ，描述這個目錄指向一個子卷。

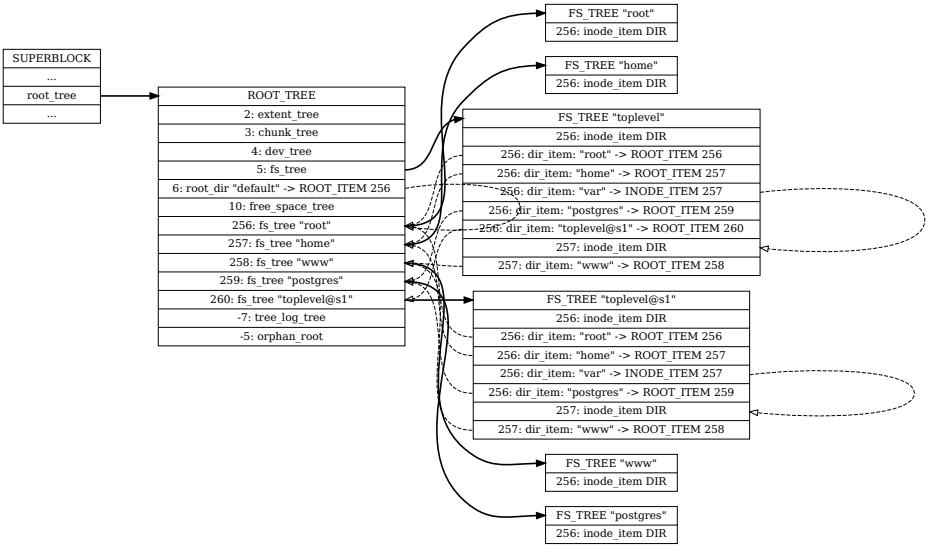
比如上圖 FS_TREE toplevel 中，有兩個對象，第一個 256 是（子卷的）根目錄，第二個 257 是 "var" 目錄，256 有4個子目錄，其中 "root" "home" "postgres"

這三個指向了 ROOT_TREE 中的對應子卷，而 "var" 指向了 inode 257 。然後 257 有一個子目錄叫 "www" 它指向了 ROOT_TREE 中 object_id 爲 258 的子卷。

以上是子卷、目錄、 inode 在 btrfs 中的記錄方式，你可能想知道，如何記錄一個快照呢？ 如果我們在上面的佈局基礎上執行：

```
btrfs subvolume snapshot toplevel t
oplevel/toplevel@s1
```

那麼產生的數據結構大概如下所示：


SUPERBLOCK
...
root_tree
...
ROOT_TREE
2: extent_tree
3: chunk_tree
4: dev_tree
5: fs_tree
6: root_dir "default" -> ROOT_ITEM 256
10: free_space_tree
256: fs_tree "root"
257: fs_tree "home"
258: fs_tree "www"
259: fs_tree "postgres"
260: fs_tree "toplevel@s1"
-7: tree_log_tree
-5: orphan_root
FS_TREE "root"
256: inode_item DIR
FS_TREE "home"
256: inode_item DIR
FS_TREE "toplevel"
256: inode_item DIR
256: dir_item: "root" -> ROOT_ITEM 256
256: dir_item: "home" -> ROOT_ITEM 257
256: dir_item: "var" -> INODE_ITEM 257
256: dir_item: "postgres" -> ROOT_ITEM 259
256: dir_item: "toplevel@s1" -> ROOT_ITEM 260
257: inode_item DIR
257: dir_item: "www" -> ROOT_ITEM 258
FS_TREE "toplevel@s1"
256: inode_item DIR
256: dir_item: "root" -> ROOT_ITEM 256
256: dir_item: "home" -> ROOT_ITEM 257
256: dir_item: "var" -> INODE_ITEM 257
256: dir_item: "postgres" -> ROOT_ITEM 259
257: inode_item DIR
257: dir_item: "www" -> ROOT_ITEM 258
FS_TREE "www"
256: inode_item DIR
FS_TREE "postgres"
256: inode_item DIR

在 ROOT_TREE 中增加了 260 號子卷，其內容複製自 toplevel 子卷，然後 FS_TREE toplevel 的 256 號 inode 也就是根目錄中增加一個 dir_item 名叫 "toplevel@s1" 它指向 ROOT_ITEM 的 260 號子卷。這裏看似是完整複製了整個 FS_TREE 的內容，這是因爲 CoW b-tree ，當只有一個葉子時就複製整個葉子。如果子卷內容再多一些，除了葉子之外還有中間節點， 那麼只有被修改的葉子和其上的中間節點需要複製。

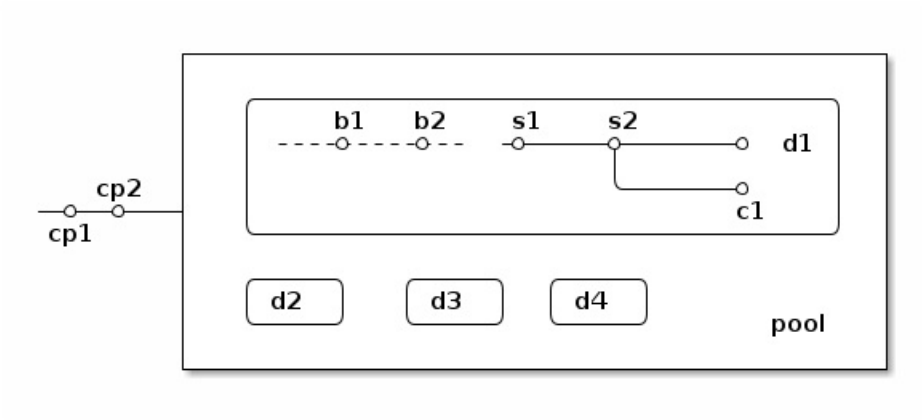
從子卷和快照的這種實現方式，可以看出：**雖然子卷可以嵌套子卷，但是對含有嵌套子卷的子卷做快照難以快速實現**。因此在目前實現的 btrfs 語義中，當子卷 S1 嵌套有別的子卷 S2 的時候，對 S1 做

ZFS 的數據集（dataset）、快照（snapshot）、克隆（clone）、書籤（bookmark）和檢查點（checkpoint）

Btrfs 給傳統文件系統只增加了子卷的概念，相比之下 ZFS 中類似子卷的概念有好幾個，分別叫：

- 數據集（dataset）
- 快照（snapshot）
- 克隆（clone）
- 書籤（bookmark）：從 ZFS on Linux v0.6.4 開始
- 檢查點（checkpoint）：從 ZFS on Linux v0.8.0 開始

梳理一下這些概念之間的關係也是最初想寫下這篇筆記的初衷。先畫個簡圖，隨後逐一講講這些東西：



數據集（dataset）

先從最簡單的概念說起。在 ZFS 的術語中，把底層管理和釋放存儲設備空間的叫做 ZFS 存儲池（pool），簡稱 zpool，其上可以創建多個數據集（dataset）。容

易看出數據集的概念直接對應 btrfs 中的子卷。也有很多介紹 ZFS 的文檔中把一個數據集（dataset）叫做一個文件系統（filesystem），這或許是想要和（像 Solaris 的 SVM 或者 Linux 的 LVM 這樣的）傳統的卷管理器與其上創建的多個文件系統（Solaris UFS 或者 Linux ext）這樣的上下層級做類比。從 btrfs 的子卷在內部結構中叫作 FS_TREE 這一點可以看出，至少在 btrfs 早期設計中大概也是把子卷稱爲 filesystem 做過類似的類比的。

與 btrfs 的子卷不同的是，ZFS 的數據集之間是完全隔離的，（除了後文會講的 dedup 方式之外）不可以共享任何數據或者元數據。