Overhead nell'allocazione dei blocchi

Creazione delle partizioni (il *prompt* `#` indica che il comando va eseguito con un'utenza dotata dei privilegi d'amministrazione, che, per esempio, possono essere acquisiti con il comando sudo –s): `fdisk` inizia una sessione interattiva in cui è possibile determinare i dati da scrivere nella tabella delle partizioni del disco.

(Questi appunti presuppongono una certa dimestichezza con la *shell* e alcuni comandi per la manipolazione del *file system*. All'inizio è istruttivo consultare il manuale di ciascuno: per esempio, `man fdisk`. Inoltre può essere estremamente utile consultare il sito `http://explainshell.com/` per capire a fondo le parti che compongono ciascun comando).

```
# fdisk /dev/sda
```

Creazione del file system (di tipo EXT2) sulla prima partizione.

```
# mkfs.ext2 /dev/sda1
```

Il programma `mkfs.ext2` sceglie la dimensione del blocco secondo alcune euristiche studiate per ottimizzare il bilanciamento fra efficienza e *overhead*. Per forzare la scelta della dimensione 1024:

```
# mkfs.ext2 -b 1024 /dev/sda1
```

Con il programma `dumpe2fs` è possibile esplorare la struttura dati per la gestione del *file system*.

```
# dumpe2fs /dev/sda1 | less
```

Perché il *file system* possa essere utilizzato *deve essere "montato"*, occorre cioè che il sotto-albero (in realtà in generale un DAG) del nuovo *file system* sia applicato a una foglia dell'albero principale (*root file system*). Una directory spesso usata a questo scopo è `/mnt`.

```
# mount /dev/sda1 /mnt
```

Ora il file system sarà accessibile a partire dalla directory `/mnt` (*mount point*); con `dumpe2fs` è possibile vedere che nella struttura dati il *file system* è indicato come "*not clean*": tornerà "*clean*" solo con un'operazione di `umount` (automatica durante lo *shutdown* del sistema).

Per studiare l'occupazione in blocchi di un *file* registriamo il numero di blocchi liberi all'inizio. Lo schema da tener presente è rappresentato in fig. 1.1

```
# dump2efs /dev/sda1 2>/dev/null | grep '^Free blocks'
Free blocks: 4123156
```

Dopo ogni operazione di scrittura è bene dare un comando `sync` che garantisce che il sistema svuoti tutti i *buffer* in memoria centrale utilizzati per evitare le costose scritture sul disco.

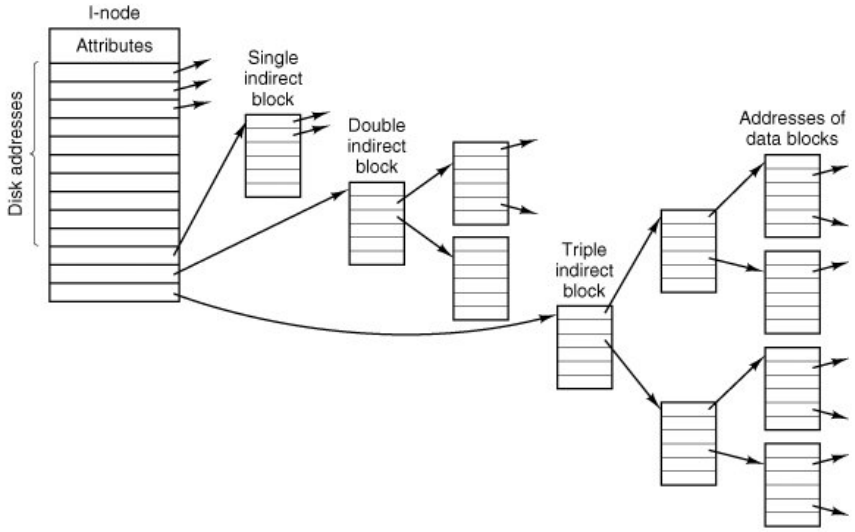

Figura 1.1: Struttura dati ad I-node

Creiamo un *file* di dimensione 0 (in pratica c'è solo l'*i-node* necessario a conservare i metadati). Si noti che in EXT2 una parte dello spazio per gli *i-node* risulta già allocata dalla creazione del *file system* (con `dumpe2fs`: per esempio potrebbero esserci 262133 "*Free inodes*"): le "zone" vengono però allocate solo quando la dimensione del file è > 0.

```
1 $ touch /mnt/pippo
2 # sync; dump2efs /dev/sda1 2>/dev/null | grep '^Free'
3 Free blocks: 4123156
4 Free inodes: 262132
5 $ stat /mnt/pippo
```

Con `stat` possiamo vedere molti dei dati contenuti nell'*i-node*: il *file* è correttamente identificato come un *file* vuoto: in questo caso `stat` riporta un uso di 0 blocchi, anche se, in effetti, alcuni *byte* sono comunque utilizzati per conservare l'*i-node*.

Copiamo alcuni *byte* nel file appena creato con il programma `dd`: li prendiamo dal *device* speciale "zero" che produce uno *stream* di *byte* 0; il parametro `bs` dice a `dd` di leggere un blocco da 1 *byte*, variando il parametro `count` possiamo decidere quanti blocchi copiare (la copia ricomincia ogni volta dall'inizio del file destinazione, a meno che non venga specificato un parametro `seek`).

```
1 $ dd if=/dev/zero of=/mnt/pippo bs=1 count=1
2 # sync; dump2efs /dev/sda1 2>/dev/null | grep '^Free'
3 Free blocks: 4123155
4 Free inodes: 262132
5 $ stat /mnt/pippo
```

La creazione di un *byte* di dati alloca un nuovo blocco. Poiché un blocco può contenere fino a 1024 *byte* ("*Block size*: 1024") non servono altri blocchi fino a 1024 *byte* di dati. `stat` riporta il fatto che per il *file* si stanno ora usando 2 blocchi: non è un errore si

tratta della stessa parola "blocco" usato con un significato diverso. Infatti molti comandi (`stat`, ma anche `du`) usano "blocco" come unità di misura (= 512 byte) dell'occupazione, contando peraltro solo i blocchi di dati. L'invariante valido dopo tutte le operazioni è:

$$\text{blocchi dati} + \text{blocchi overhead} + \text{blocchi liberi} = \text{blocchi inizialmente liberi} = 4123156$$

```
$ dd if=/dev/zero of=/mnt/pippo bs=1 count=1024
# sync; dump2efs /dev/sda1 2>/dev/null | grep '^Free'
Free blocks: 4123155
Free inodes: 262132
$ stat /mnt/pippo
```

Il numero di blocchi occupati aumenta di uno a 1025, 2049, 1025 3073, 4097, 5121, 6145, 7169, 8193, 9217, 10241, 11265.

```
$ dd if=/dev/zero of=/mnt/pippo bs=1 count=1025
# sync; dump2efs /dev/sda1 2>/dev/null | grep '^Free'
Free blocks: 4123154
Free inodes: 262132
$ stat /mnt/pippo
```

```
$ dd if=/dev/zero of=/mnt/pippo bs=1 count=11265
# sync; dump2efs /dev/sda1 2>/dev/null | grep '^Free'
Free blocks: 4123144
Free inodes: 262132
$ stat /mnt/pippo
```

A 12289 ($= 1024 \cdot 12 + 1$) abbiamo una nuova discontinuità, perché le 12 "zone" dell'*i-node* non bastano piú per indirizzare i blocchi di dati e serve quindi un nuovo blocco di *overhead*.

```
$ dd if=/dev/zero of=/mnt/pippo bs=1 count=12289
# sync; dump2efs /dev/sda1 2>/dev/null | grep '^Free'
Free blocks: 4123142
Free inodes: 262132
$ stat /mnt/pippo
```

Il nuovo blocco di *overhead* permette di allocare (indirettamente) altri 256 blocchi di dati (con indirizzi da 32 bit, infatti $\frac{1024 \cdot 8}{32} = 256$). A 274433 ($= 1024 \cdot (256 + 12) + 1$) occorre usare le doppie indirettezze (infatti il numero di blocchi utilizzati aumenta di $2 + 1 = 3$).

```
$ dd if=/dev/zero of=/mnt/pippo bs=1 count=274433
# sync; dump2efs /dev/sda1 2>/dev/null | grep '^Free'
Free blocks: 4122884
Free inodes: 262132
$ stat /mnt/pippo
```

1.1 Domande

1. Quando sarà la prossima discontinuità?

2. Qual è il numero massimo di blocchi allocabile per un file?
3. Quanti blocchi sono necessari per un file di $512000\,byte$?

1.2 Risposte

1. $1024 \cdot (256 \cdot 256 + 256 + 12) + 1 = 67383297$ (tripla indirettezza, il numero di blocchi aumenti di $3 + 1 = 4$).
2. $1024 \cdot (256^3 + 256^2 + 256 + 12) + 1 = 17247252481 \approx 16\ GiB$
3. Ne serviranno necessariamente almeno $\frac{1024000}{1024} = 1000$ per conservare i dati. A questi vanno aggiunti 5 di *overhead*: 0 per i 12 blocchi diretti, 1 per i 256 blocchi indiretti e $1 + \lceil \frac{1000-256-12}{256} \rceil = 4$ per i blocchi doppiamente indiretti (oltre naturalmente allo spazio usato per l'*i-node*).