Zettabyte File System

Una breve presentazione

Trentin Patrick

Università di Verona

14 Gennaio 2011

Contatti:
id084071@studenti.univr.it

File System (1/2)

Apparentemente la scelta del **file system** non ha molta influenza diretta sulle nostre vite.

File System

software che agevole la **manipolazione** ed **organizzazione** dei **dati** memorizzati sul disco.

Ne esistono un gran numero:

- Ext, Ext2, Ext3, Ext4;
- Fat, Fat32;
- FFS;
- HFS, HFS+;
- JFS;
- Ntfs;
- ReiserFS, Reiser4;
- UFS, UFS2;

Perché ve ne sono così tanti, e perché **aggiungerne** un altro alla lista?

File System (2/2)

Alcuni problemi dei file system tradizionali:

- Complessa **gestione** (partizioni, volumi, aggiunta e rimozione dischi);
- Non hanno difese contro la **corruzione** silente dei dati;
- Limiti dimensionali non ampi (numero di file/directory, dimensioni file);

Riflessione:

A quanti è mai capitato di perdere i propri dati, per un'errata gestione od un errore nei dischi?

In ambienti di produzione, queste situazioni non sono infrequenti!

File System (2/2)

Alcuni problemi dei file system tradizionali:

- Complessa **gestione** (partizioni, volumi, aggiunta e rimozione dischi);
- Non hanno difese contro la **corruzione** silente dei dati;
- Limiti dimensionali non ampi (numero di file/directory, dimensioni file);

Lo **Zettabyte File System** (*ZFS*) nasce per rispondere a mutate esigenze:

- Affidabilità;
- Sicurezza;
- Astrazione;
- Semplicità;

Zettabyte File System

ZFS viene sviluppato presso la **Sun Microsystem** a partire dal 2000.

In soli 6 anni ed 80.000 righe di codice, viene rimodellato il concetto di file system, secondo le parole del capo progettuale *Jeff Bonwick*:

> *Abbiamo gettato via 20 anni di tecnologie ormai superate che erano basate su assunzioni al giorno d'oggi non più vere.*

Rilasciato in **Solaris** nel 2005, ha subito ulteriore sviluppo e maturazione.

Ora è proprietà della **Oracle Corporation**, sviluppatrice di un progetto concorrente, il **Butter File System**.

Scaletta dei contenuti

1. Semplicità di gestione
   - Storage Pool e Volume Managing;
2. Astrazione della memoria
   - Dataset, strutture dati ed organizzazione dei dispositivi di memoria;
3. Affidabilità di funzionamento
   - Transazione completa, allocazione e free dei blocchi;
4. Sicurezza sui contenuti
   - Integrità end-to-end, strategie di ridondanza dati;

Scaletta dei contenuti

1. Semplicità di gestione
   - Storage Pool e Volume Managing;
2. Astrazione della memoria
   - Dataset, strutture dati ed organizzazione dei dispositivi di memoria;
3. Affidabilità di funzionamento
   - Transazione completa, allocazione e free dei blocchi;
4. Sicurezza sui contenuti
   - Integrità end-to-end, strategie di ridondanza dati;

Volume Managing vs. Storage Pool (1/2)

I file system tradizionali sono progettati per lavorare su un **device singolo**. Reso necessario il

Logical Volume Management (*LVM*)

software che gestisce direttamente i dischi, astraendo lo spazio di memorizzazione su cui lavora il file system.

Vantaggi:

- Raid, mirroring e striping emulate via software;
- Partizioni dinamiche;
- Maggiore capienza;

Svantaggi:

- Basso utilizzo parallelismo in lettura e scrittura;
- Limitata interoperabilità tra FS e LVM;
- Possibile causa di problemi di inconsistenza su disco;

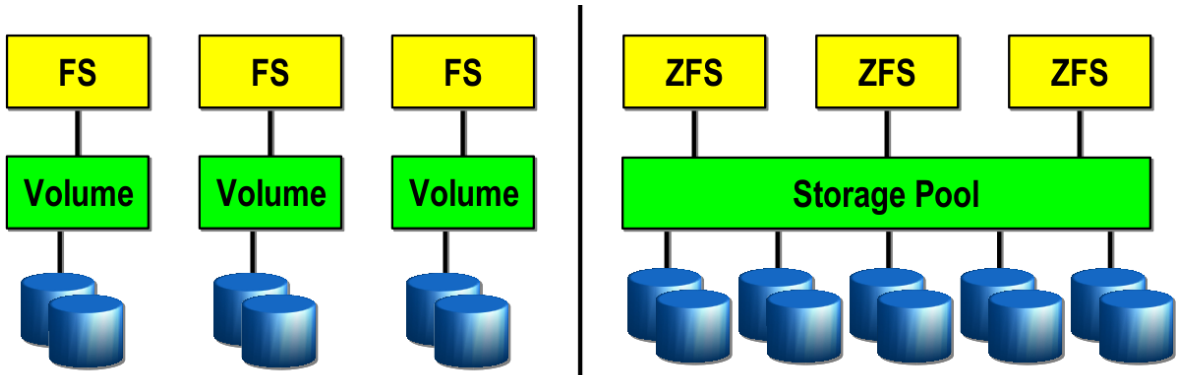
Volume Managing vs. Storage Pool (2/2)

Storage Pool (*zpool*)

entità virtuale che raccoglie più **file system** logicamente indipendenti tra loro, appoggiandosi ad uno spazio di **memoria comune**.

Vantaggi:

- Lettura/scrittura effettuate in **parallelo**;
- Ogni FS può espandersi dinamicamente;
- Supporto per *boot* e *swapping* di memoria;
- Possibile aggiungere e rimuovere dischi dal pool **senza smontare** il file system;
- Supporto per Raid-Z, striping e mirroring;



File System dei grandi numeri

ZFS lavora a **128 bit**, ciò garantisce una capacità di memorizzazione pressoché infinita:

- 2^{48} file per directory;
- 2^{64} file system per zpool;
- 2^{64} dischi rigidi per zpool, e zpool in un sistema;
- 2^{78} bytes dimensione massima di ogni zpool;
- 2^{64} bytes dimensione massima di file, file system ed attributi;

$$2^{64} \equiv 18.000.000.000.000.000.000$$

Creando mille file al secondo, occorrerebbero 9.000 anni per saturare le capacità del sistema. Tutte l'energia necessaria per bollire gli oceani non sarebbe sufficiente!

Scaletta dei contenuti

1. Semplicità di gestione
   - Storage Pool e Volume Managing;
2. Astrazione della memoria
   - Dataset, strutture dati ed organizzazione dei dispositivi di memoria;
3. Affidabilità di funzionamento
   - Transazione completa, allocazione e free dei blocchi;
4. Sicurezza sui contenuti
   - Integrità end-to-end, strategie di ridondanza dati;

Vista astratta: Dataset (1/2)

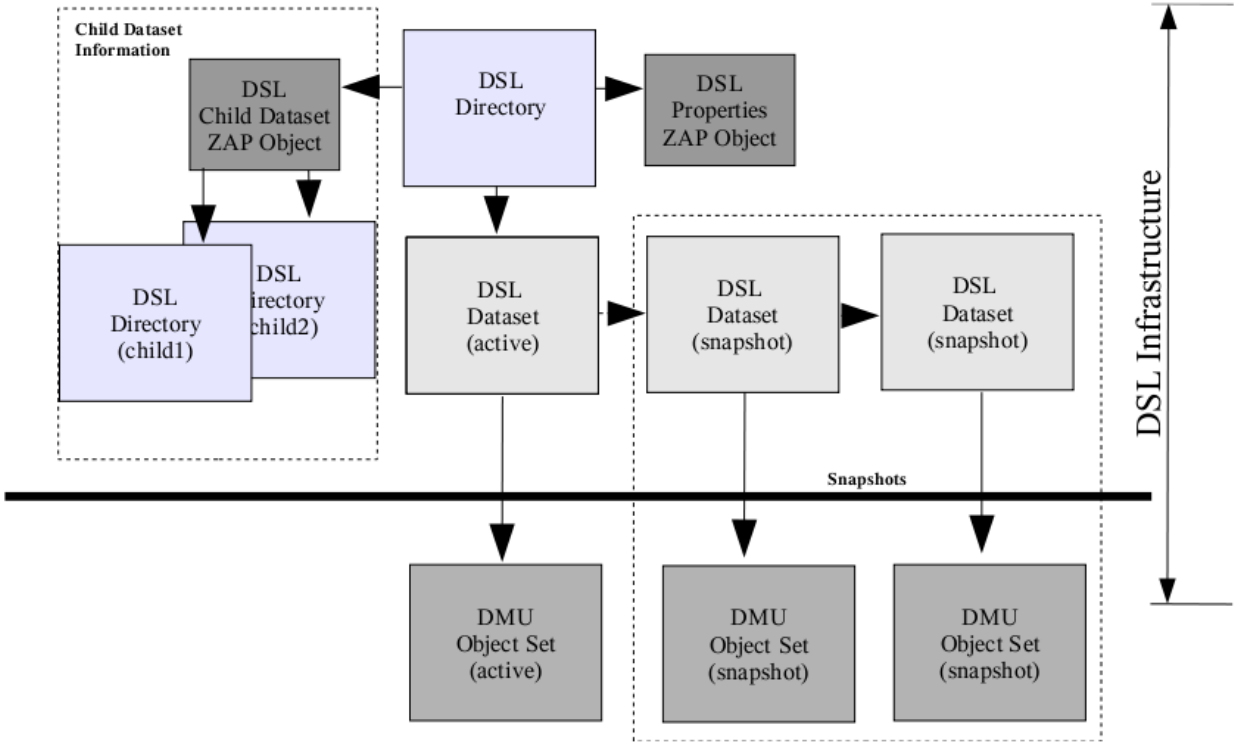
Dataset

In ZFS è un **insieme di oggetti** correlati tra loro, dotati di **proprietà comuni** ereditarie.

Esistono quattro tipologie di **dataset**:

- **file system**: memorizza ed organizza oggetti accessibili mediante la semantica *POSIX*;
- **snapshot**: istantanea in sola lettura di un particolare file system, clone o volume;
- **clone**: dataset originato da uno snapshot accessibile in scrittura, supporta le operazioni disponibili sul file system;
- **volume**: volume logico di ZFS esportato come device a blocchi (*Es: usb stick*).

Vista astratta: Dataset (2/2)



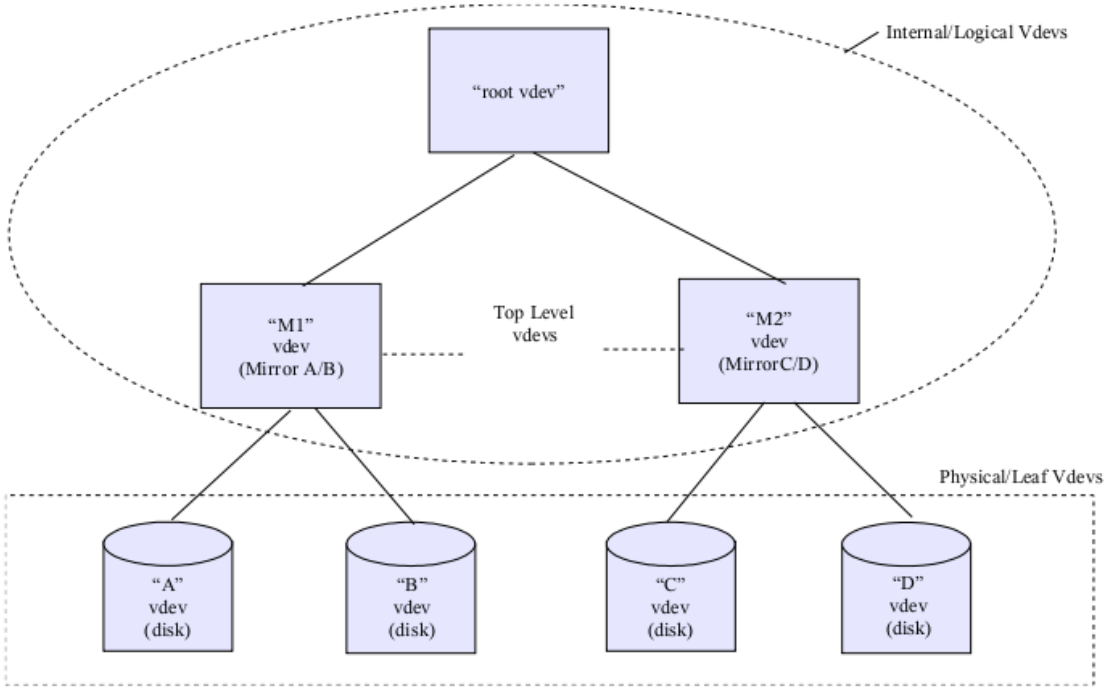
Il **dataset** attivo viene anche chiamato *live file system*.

Nota: Esistono relazioni di interdipendenza molto complesse tra i dataset!

Vista fisica (1/5): zpool

I **dischi** di un pool sono organizzati con una struttura ad **albero**:

- ogni foglia descrive un disco fisico preciso del pool (*leaf vdev*);
- ogni nodo interno è un'entità virtuale che raccoglie più device fisici e ne astrae le proprietà (*logical virtual vdev*).

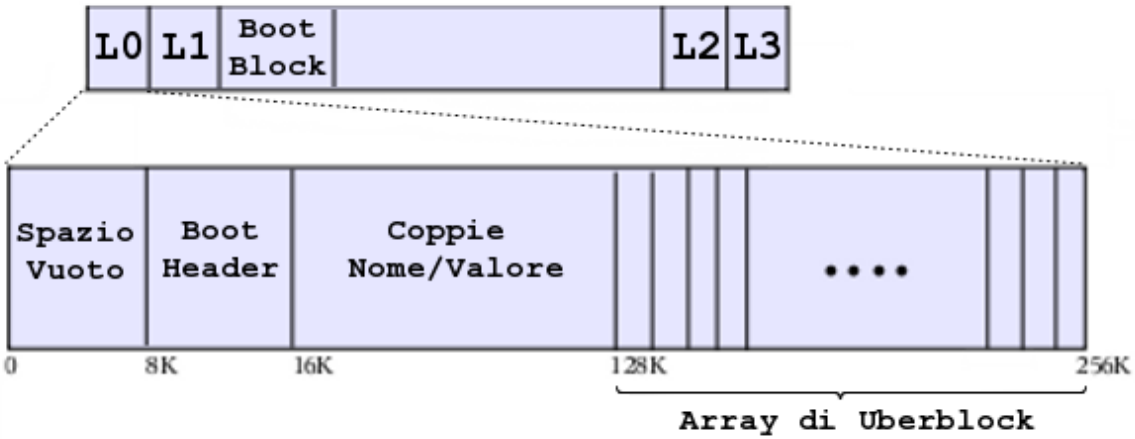


Vista fisica (2/5): vdev label

Vdev Label

identifica e **descrive un device** fisico, e tutti gli altri device che condividono uno stesso **vdev padre** nell'albero dei dispositivi.

La **vdev label** (256 Kb) è replicata in **4 copie** per disco, due all'inizio e due in fondo al disco per garantirne sempre almeno una copia disponibile.



L'importanza di queste strutture dati impone che la **vdev label** sia aggiornata con una transazione a **due stadi**.

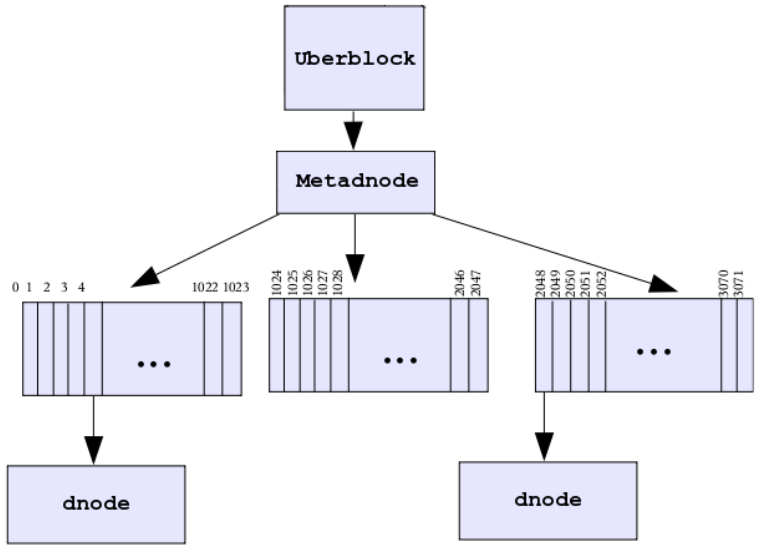
Vista fisica (3/5): uberblock

Uberblock

blocco **radice** dell'intero **albero di dati**, detiene le informazioni per l'accesso ai contenuti del pool.

In ogni istante un solo **uberblock** soddisfa entrambi i requisiti:

- ha gruppo di transazione massimo tra quelli nell'array.
- ha valore di *checksum* valido;



Ad ogni completamento di una fase di scrittura, un nuovo **uberblock** viene inserito in *round robin* nell'array della **vdev label**.

Vista fisica (4/5): dnode

dnode

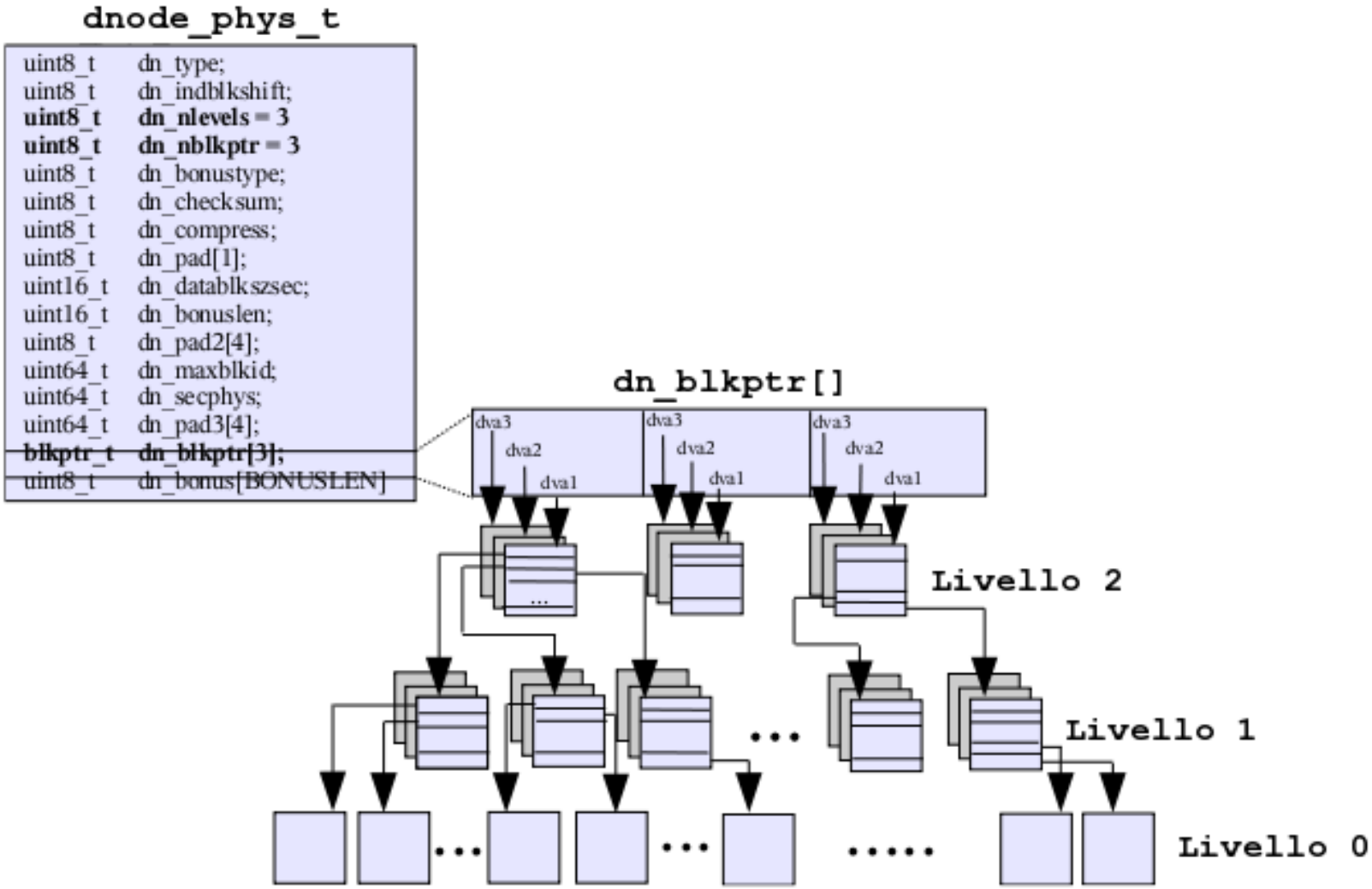
simile ad un **inode**, struttura di 512 byte che definisce ogni oggetto contenuto di uno **zpool**.

Un **dnode** possiede tre puntatori a blocchi. Ne esistono di due categorie principali:

- blocchi **dati**, di dimensione $\geq$ 512 B e $\leq$ 128 kB;
- blocchi **intermedi**, contententi 1024 puntatori. Con sei livelli di intermediazione massimi, la dimensione di un file è limitata a 2^{64} B.

Ogni **blocco** è identificato da un **id** numerico a 64 bit con il quale si può risalire alla sua esatta posizione nell'albero di blocchi.

Vista fisica (4/5): dnode



Vista fisica (5/5): gestione spazio

Lo spazio di indirizzamento di ogni disco viene scomposto in qualche centinaio di regioni virtuali dette Metaslab. A ciascuno è associato uno

Spacemap

struttura dati che riassume le operazioni di **allocazione** e **de-allocazioni** effettuate nel tempo su un particolare **metaslab**.

Vantaggi:

- Manipolabile in tempo costante per appending sull'ultimo blocco in memoria (scalabilità);
- Non necessita di inizializzazione: l'assenza di entry implica che tutto lo spazio è libero;
- Le prestazioni non degradano se il pool è quasi interamente allocato.

Dallo Spacemap si deriva un albero AVL ordinato per offset, sul quale vengono simulate le operazioni memorizzate.

Scaletta dei contenuti

1. Semplicità di gestione
   - Storage Pool e Volume Managing;
2. Astrazione della memoria
   - Dataset, strutture dati ed organizzazione dei dispositivi di memoria;
3. Affidabilità di funzionamento
   - Transazione completa, allocazione e free dei blocchi;
4. Sicurezza sui contenuti
   - Integrità end-to-end, strategie di ridondanza dati;

Striping Dinamico

I blocchi di dimensione variabile consentono un utilizzo ottimale dello **striping dinamico**.

striping

tecnica che distribuisce unità di dati - i blocchi nel caso di ZFS - su più dischi rigidi.

Vantaggi:

- incremento del *throughput* grazie al parallelismo;
- minor *frammentazione interna* grazie alla dimensione variabile dei blocchi;

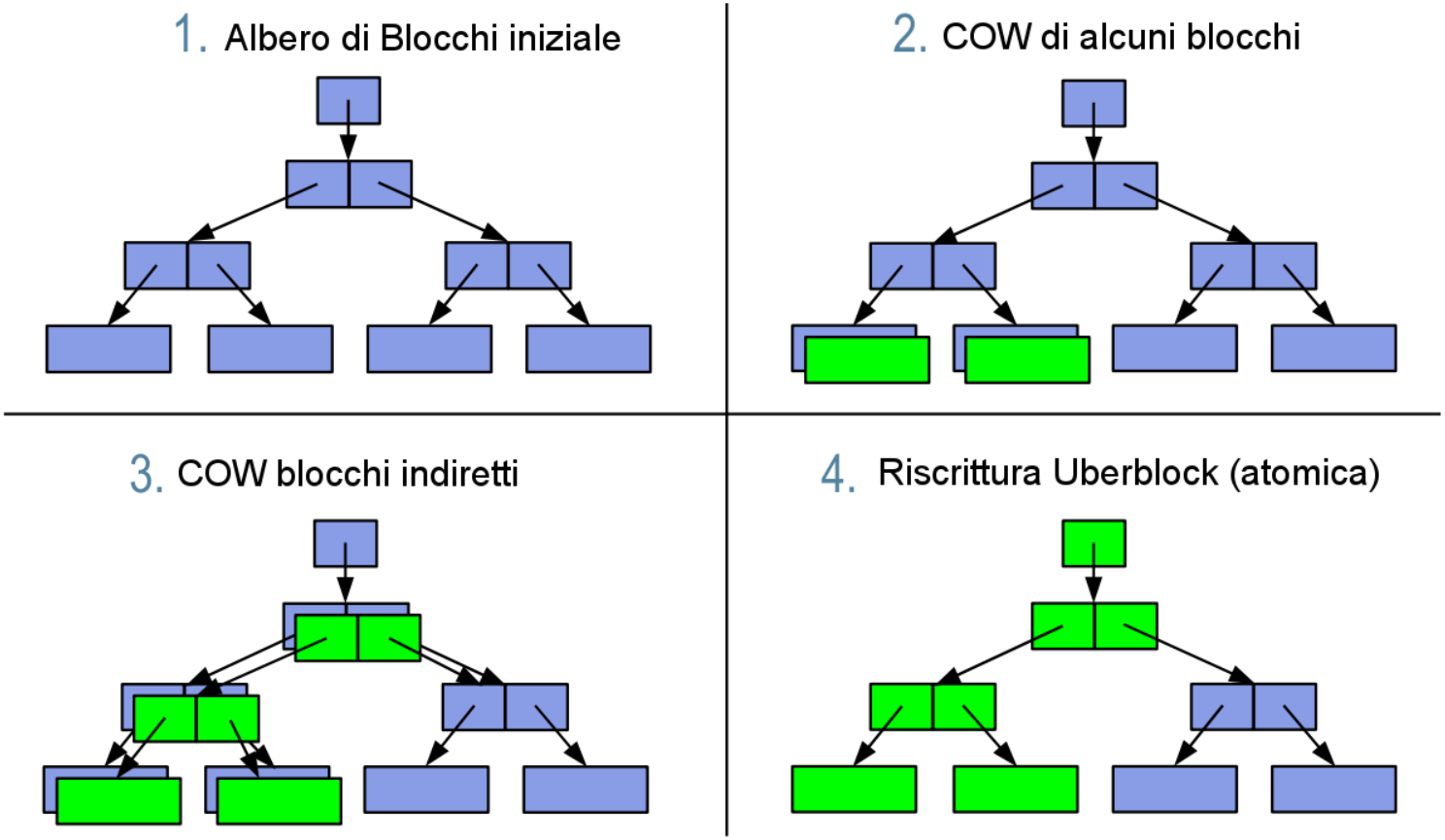
Lo **stripe** viene calcolato sulla dimensione del file rispetto al numero di dischi nello **zpool** sui quali viene distribuito.

Allocazione Blocchi

Policy di allocazione blocchi:

- Selezione **disco rigido**:
  Alternato con politica *round robin* ogni 512K di dati, per un miglior uso del meglio buffering reading e I/O sequenziale. Dischi sottoutilizzati vengono favoriti.
- Selezione **Metaslab**:
  Preferite le regioni piú esterne del disco, per un minore *seek latency*.
- Selezione **Blocco**:
  Algoritmo *first-fit* fino al raggiungimento di una certa soglia di spazio libero, poi *best-fit*.

Copy on Write (1/2)



Ogni scrittura è una **transazione** completa: principio *all or nothing*.
Ogni **transazione** alloca nuovi blocchi: NON si sovrascrivono quelli vecchi.

Copy on Write (2/2)

Una **transazione** contiene una lista di **modifiche** su **più dnode** del file system.

Vantaggi:

- File system **consistente** anche in caso di power failure;
- Possibile ottimizzazione dei gruppi di operazioni su disco;
- Semplice implementazione degli **snapshot**.

Svantaggi:

- Possibile accumulo di operazioni incompiute: necessario *Intent Log*

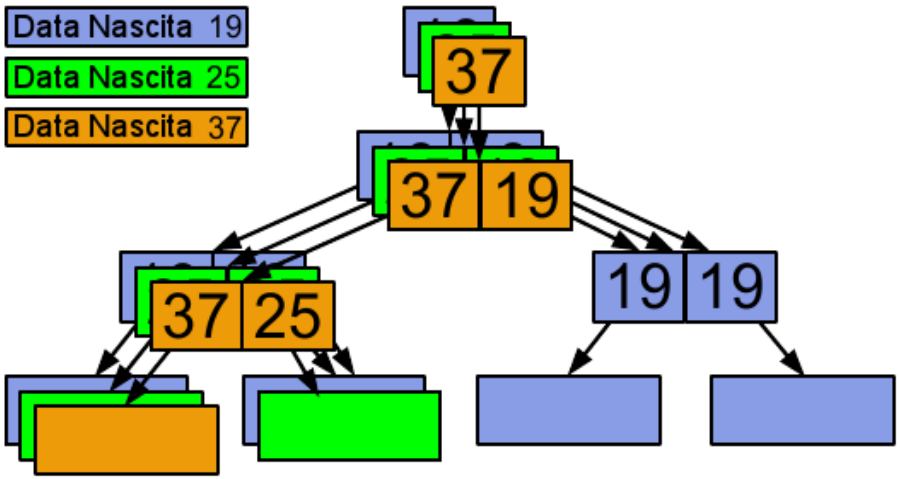
Snapshot (1/2)

Implementato utilizzando il **copy on write** e due strutture dati:

- *ID numerico* del gruppo di transazione in cui è stato creato il blocco;
- *Dead list*: puntatori a blocchi referenziati in uno snapshot precedente e non più allocati nel dataset di riferimento.

Creazione Snapshot:

Viene salvata la radice del *dataset* di riferimento, con una operazione atomica in O(1).



Spazio Occupato:

- Inizialmente nullo;
- Cresce con il numero di modifiche apportate successivamente al file system.

Snapshot (2/2)

Rimozione blocco:

- Data di nascita blocco $>$ data di nascita ultimo snapshot
  Lo **spazio liberato** viene aggiunto allo **spacemap** del metaslab a cui appartiene il blocco.
- Data di nascita blocco $\leq$ data di nascita ultimo snapshot
  Il **blocco** viene aggiunto alla *dead list* del file system senza liberarne lo spazio.

Rimozione snapshot:

Richiede un **tempo proporzionale** al numero di blocchi da liberare, $O(\Delta)$;

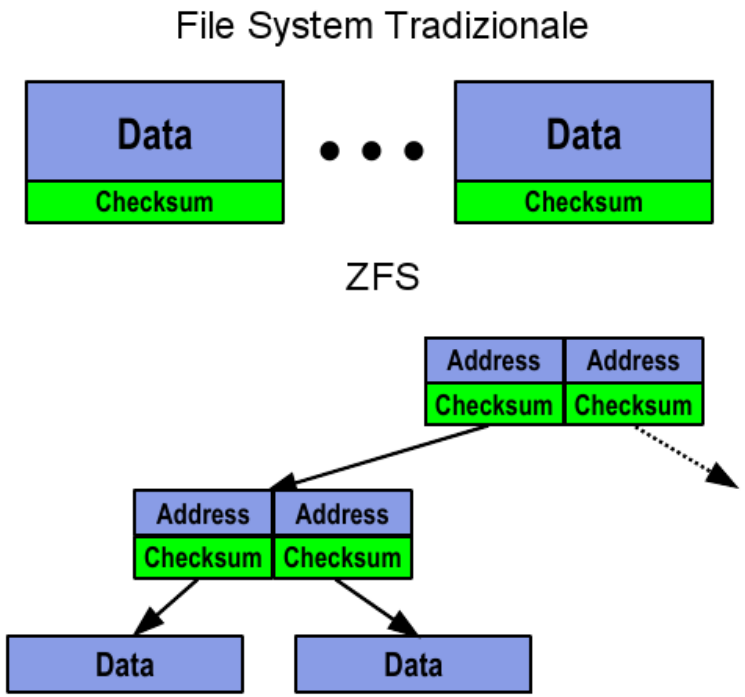
I blocchi nella **dead list** dello snapshot successivo, nati dopo quello in rimozione, vengono liberati.

Scaletta dei contenuti

1. Semplicità di gestione
   - Storage Pool e Volume Managing;
2. Astrazione della memoria
   - Dataset, strutture dati ed organizzazione dei dispositivi di memoria;
3. Affidabilità di funzionamento
   - Transazione completa, allocazione e free dei blocchi;
4. Sicurezza sui contenuti
   - Integrità end-to-end, strategie di ridondanza dati;

Integrità end-to-end

- in ZFS il **checksum** dei blocchi viene memorizzato **con i metadati** del blocco padre;
- questa separazione rende l'**albero autovalidante**;
- una incongruenza implica necessariamente che i dati sono danneggiati;
- in presenza di forme di **ridondanza**, è possibile recuperare e **ripristinare** i dati persi.



Data Scrubbing

scrubber

Software di **revisione automatica** ed online del **file system** che individua e corregge eventuali errori presenti in memoria.

Alcuni blocchi dati sono accessibili da più blocchi intermedi: rischio di verificare più volte gli stessi contenuti.

Implementazione:

- l'esecuzione dell'algoritmo di scrubbing è eseguita in background e in contesto *sync*. Priorità data ad I/O normale;
- i dataset vengono aggiunti ad una coda e visitati in ordine temporale di creazione;
- se viene incontrato un blocco con data di nascita $\leq$ a quella di un dataset giá visitato, non ne viene verificato l'intero sottoalbero.

Resilvering (1/2)

resilvering

processo che a partire da device integri ricostruisce e ripara i contenuti di un disco danneggiato.

Requisiti: qualche forma di ridondanza dati.

Approccio tradizionale:

- da un mirror: copia byte per byte di un disco;
- da un raid: xor dati stripe con informazioni di parità;
- copia dell'intero disco indipendentemente dalla struttura del file system e dalla quantità di spazio libero;
- non viene verificata la consistenza dei dati copiati o ricostruiti.

Resilvering (2/2)

Approccio ZFS:

- la copia dei blocchi viene eseguita discendendo l'albero dati dello storage pool;
- maggiore precedenza ai dati più importanti;
- tutti i contenuti vengono verificati con il rispettivo checksum prima della copia;
- risparmio di tempo se il disco è relativamente vuoto rispetto alla propria capacità;
- possibile limitazione del resilvering ai soli dati modificati nell'intervallo di tempo in cui un disco è rimasto inacessibile;

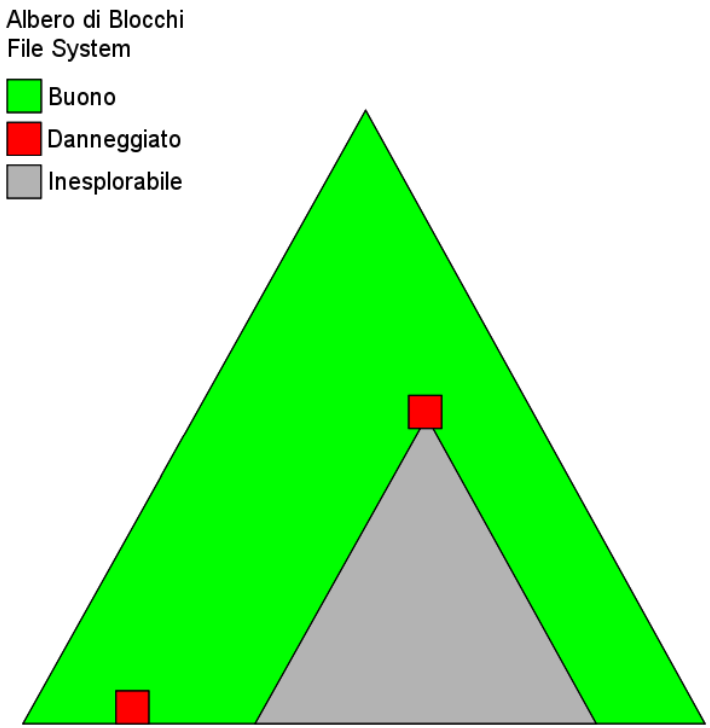
Ridondanza Dati

Un **errore** in un blocco può avere diversa gravità:

- blocco dati: lettura interrotta con un *EIO*;
- blocco intermedio: intero sotto albero di dati integri diventa inacessibile.

Necessario utilizzare forme di **ridondanza dati**:

1. Ditto Blocks;
2. Raid-Z;
3. Mirroring;



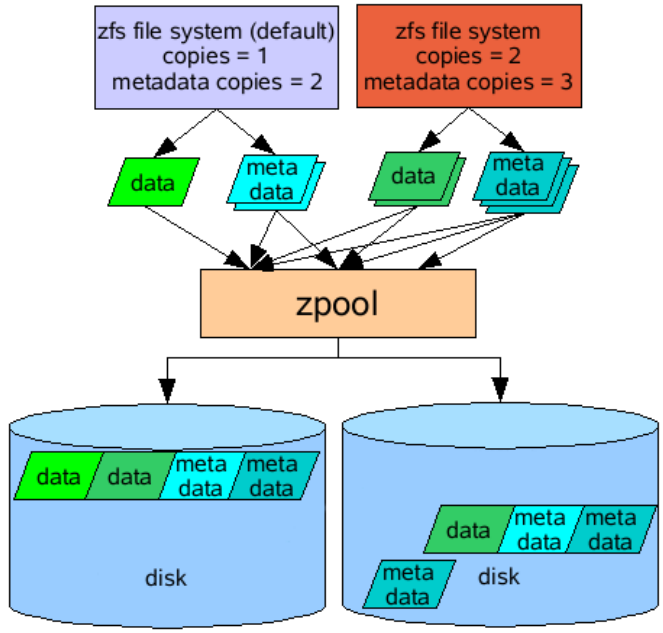
Ditto Blocks (1/2)

In ZFS ogni puntatore è una **tripla** di **indirizzi**, che puntano a blocchi dagli stessi contenuti.

Ditto Block

blocco del file system contenente una copia dei dati di un altro blocco.

- *Politica Diffusiva*
  I ditto blocks sono distribuiti quanto più possibile tra i dispositivi.
  Se è disponibile **un** solo **disco**, i blocchi ridondanti vengono distanziati di almeno **1/8** del disco.



Ditto Blocks (2/2)

- *Politica Ridondante*
  Blocchi più vicini alla directory di root del file system debbono avere più repliche.
  - due repliche: dati di accesso globale dello storage pool;
  - una replica: metadati del file system;
  - nessuna replica: dati utente.

  Minimo impatto sulle prestazioni: i blocchi intermedi dotati di replica sono meno del 2% del file system.

  È possibile ridefinire in modo personalizzato le politiche di ridondanza per ciascun file system.

Raid-Z (1/2)

Il **Raid-Z** eredita molte delle caratteristiche del **Raid-5** tradizionale, in cui:

- i blocchi e le informazioni di parità sono distribuiti su più dischi;
- il sistema è in grado di resistere alla perdita di un disco;
- il file system è vulnerabile al **write hole**.

write hole

È un problema di **corruzione silente dei dati** che si verifica quando per un guasto o una perdita di corrente le informazioni di **parità non** vengono **aggiornate**. Se ciò non viene rilevato prima di un *data error* sullo stripe, verranno ricostruiti i dati mancanti con le informazioni inconsistenti di parità.

Raid-Z (2/2)

Il **Raid-Z** è immune al problema del **write hole**:

La dimensione variabile dei blocchi garantisce che ad ogni fase di scrittura si concluda con uno stripe completo.

La strategia **copy on write** garantisce che ogni fase di scrittura si concluda in modo consistente.

LBA \ Disk	A	B	C	D	E
0	P_0	D_0	D_2	D_4	D_6
1	P_1	D_1	D_3	D_5	D_7
2	P_0	D_0	D_1	D_2	P_0
3	D_0	D_1	D_2	P_0	D_0
4	P_0	D_0	D_4	D_8	D_{11}
5	P_1	D_1	D_5	D_9	D_{12}
6	P_2	D_2	D_6	D_{10}	D_{13}
7	P_3	D_3	D_7	P_0	D_0
8	D_1	D_2	D_3	X	P_0
9	D_0	D_1	X	P_0	D_0
10	D_3	D_6	D_9	P_1	D_1
11	D_4	D_7	D_{10}	P_2	D_2
12	D_5	D_8	•	•	•

Conclusioni

Abbiamo visto come con ZFS sia stata rivalutata l'importanza di avere un file system

- Sicuro
- Affidabile

esplorato alcune le soluzioni che sono state adottate

- Transazionalità
- Verifica integrità dei singoli oggetti
- Ridondanza

Ci sono domande?