ZFS in Linux

21/09/2025
v1.05

I filesystem, in quanto responsabili dell'organizzazione e della gestione dei dati nei dispositivi di archiviazione, sono componenti essenziali di qualsiasi sistema operativo. Essi forniscono un livello di astrazione che semplifica l'interazione delle applicazioni con file e directory, nascondendo la complessità della gestione fisica dei dati e prevenendo conflitti di risorse tra i processi. Nel corso del tempo, i filesystem si sono evoluti per diventare sempre più sofisticati ed efficienti. In ambiente Linux, tra i filesystem più utilizzati figurano EXT4, XFS, Btrfs e ZFS. Quest'ultimo si distingue per le sue avanzate funzionalità, tra cui la garanzia di integrità dei dati, la gestione efficiente di grandi volumi di storage e altre caratteristiche di livello enterprise, che ne fanno una scelta privilegiata in ambito server, in sistemi di backup e piattaforme per l'archiviazione affidabile dei dati.

Il presente documento è il risultato delle conoscenze ed esperienze dell'autore. Non sono stati impiegati modelli linguistici di grandi dimensioni (LLM) né altri strumenti di intelligenza artificiale nella sua stesura.

Questo documento è redatto in LaTeX (distribuzione TeX Live 2022 o successiva).

Licenza

Avvertenze

Questo documento è un "*work in progress*". Sebbene mi sia adoperato per far sì che le informazioni contenute in esso siano corrette, non posso garantire che tali informazioni siano complete o esenti da errori. Questo documento e i suoi contenuti vengono dunque forniti "*AS IS*", cioè nello stato in cui si trovano, senza garanzia alcuna, espressa o implicita, pertanto chiunque utilizzi questo documento e i suoi contenuti lo fa a proprio rischio. Personalmente declino ogni responsabilità per eventuali danni, diretti, indiretti, incidentali o consequenziali derivanti dall'uso, corretto o scorretto, delle informazioni contenute nel presente documento. Non sono inoltre responsabile di eventuali errori o inesattezze presenti nelle fonti esterne citate (link a piè di pagina, riferimenti utili, pagine man) né posso garantirne la disponibilità continua.

Chiunque voglia contribuire alla stesura di tale documento o semplicemente segnalarmi errori, omissioni, inesattezze o possibili miglioramenti, può contattarmi all'indirizzo e-mail "*flavio.cappelli@gmail.com*".

Tale indirizzo e-mail è da ritenersi non pubblico, ed è pertanto esplicitamente vietata la sua inclusione in archivi di indirizzi, liste di distribuzione o altre forme di trattamento dati, senza la mia esplicita autorizzazione scritta. È inoltre vietato inviarmi pubblicità non espressamente richiesta.

Indice

1 Introduzione

ZFS (Zettabyte File System)[1] è un filesystem open source a 128bit, sviluppato in origine da Sun Microsystems (ora Oracle) per il sistema operativo Solaris e in seguito portato su OpenSolaris, FreeBSD, NetBSD, MacOS e Linux. ZFS è molto versatile e le sue molteplici caratteristiche derivano dall'integrazione, in un unico prodotto, di diversi concetti mutuati da vari altri filesystem (capacità di memorizzazione praticamente infinita, scritture raggruppate in transazioni, strategia copy-on-write, integrità dei dati mediante checksum e self-healing, blocchi di dimensioni variabili, compressione trasparente, encryption, logical volume management, dataset, snapshot, cloning, resilvering intelligente, supporto di device di cache, gestione delle quote, compatibilità POSIX, amministrazione intuitiva, portabilità, ...). ZFS è un filesystem di classe enterprise, che da il meglio di se in sistemi con memoria ECC e storage professionale composto da molteplici array di dischi, ma ciò non significa che non possa essere utilizzato su hardware "casalingo". Per ottenere delle prestazioni accettabili sono però necessari 1GB di RAM per TB di storage (oltre la RAM necessaria per il sistema operativo e le applicazioni) e una CPU multi-core recente. Come vedremo, è inoltre fondamentale effettuare il "tuning" del filesystem, in funzione della tipologia di dati memorizzati (ZFS non è "plug and play" come EXT4 o XFS). Notare che ZFS non ambisce ad essere il filesystem più veloce: il suo obiettivo primario è l'integrità dei dati. Ma nessun altro filesystem è in grado di mettere a disposizione tutte le caratteristiche che offre ZFS.

ZFS è rilasciato con licenza CDDL, che è incompatibile con la GPL, per cui il suo codice non può essere integrato direttamente nel kernel Linux, ma deve essere distribuito come sorgente, o come binario separato dal kernel (allineato alla versione di kernel considerata). Molte distribuzioni Linux forniscono nei loro repository un modulo "zfs" precompilato per ogni versione di kernel supportata.

2 Zpool, vdev e dataset

Prima di vedere i comandi principali per operare con il filesystem ZFS occorre comprendere i concetti di "zpool" (o più semplicemente "pool"), "vdev" e "dataset".

ZFS si basa essenzialmente su una collezione ("pool") di uno o più storage virtuali ("vdev"). Un vdev è costituito da uno (e solo uno) dei seguenti insiemi di dispositivi fisici:

- dischi singoli;
- vdev ridondanti o MIRROR (noti in altri ambiti come RAID1);
- vdev con parità singola o RAIDZ1 (noti in altri ambiti come RAID5);
- vdev con parità doppia o RAIDZ2 (noti in altri ambiti come RAID6);
- vdev con parità tripla o RAIDZ3 (noti in altri ambiti come RAID7).

Le scritture vengono distribuite tra i vdev del pool su una base relativamente uniforme. All'interno dei vdev è attiva la ridondanza mediante "mirroring o parity" (se vengono utilizzati due o più dischi per vdev)[2] mentre tra i vdev del pool è sempre attivo il "data striping"[3]. Lo spazio complessivo, fornito dai vdev, è reso disponibile a tutti gli oggetti del pool ("dataset"). Si possono avere quattro tipologie di "dataset":

ZFS Filesystem	Classica organizzazione dei dati in file e cartelle, con accesso mediante semantica POSIX;
ZFS Volume	Volume logico ("zvol") esportato da ZFS come "device a blocchi" in "*/dev/zvol/*";
ZFS Snapshot	Istantanea non modificabile di un particolare filesystem o volume (anche quelli clonati, vedere tipologia seguente);
ZFS Clone	Copia di uno snapshot (filesystem o volume) che è possibile modificare.

[1] Vedere https://it.wikipedia.org/wiki/ZFS_(file_system)
[2] Vedere https://it.wikipedia.org/wiki/RAID
[3] Vedere https://en.wikipedia.org/wiki/Data_striping

I dataset sono strutturati secondo una gerarchia e ciascuno eredita alcune proprietà dal genitore (che eventualmente possono essere modificate). Ad esempio, è possibile restringere lo spazio concesso ad un dataset e ai suoi figli, forzare le scritture ad essere sincrone, abilitare la compressione e/o la crittografia, ecc.

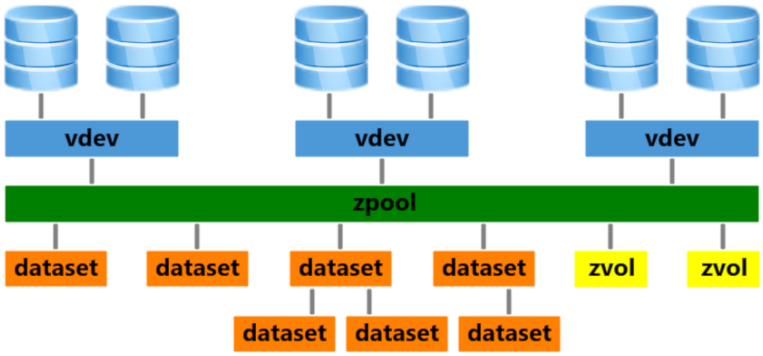


Figura 1: Esempio di filesystem ZFS

Sebbene ZFS offra molti vantaggi, ci sono alcuni fattori che occorre considerare nel suo impiego:

- Al 90% di occupazione della capacità di storage, ZFS cambia la metodologia di ottimizzazione, passando da una strategia focalizzata sulle prestazioni a una orientata al risparmio dello spazio, con conseguente rallentamento delle operazioni. Per massimizzare le prestazioni di I/O e prevenire complicazioni nella sostituzione dei dischi, è bene aggiungere più capacità prima che il pool superi l'80% di spazio occupato.
- Nel valutare il numero di dischi da utilizzare per vdev, occorre considerare la dimensione delle unità di storage e la quantità di tempo richiesta per il "resilvering" (processo di ricostruzione del vdev, con recupero dei dati mancanti a partire dai dati di parità o ridondanza). Maggiore è la dimensione del vdev, più lungo sarà il tempo di resilvering; inoltre, poiché tale operazione è piuttosto intensiva (specie per il generico RAIDZ) è possibile che prima del completamento anche un altro disco smetta di funzionare, con conseguente perdita del pool qualora non restino sufficienti informazioni di parità. Per questo motivo, lo schema RAIDZ1 viene sconsigliato per unità di dimensioni superiori a 1 TB.
- Il MIRROR sfrutta in modo meno efficiente lo spazio sui dischi ma, come vedremo, fornisce prestazioni migliori rispetto al generico RAIDZ, specie con letture e scritture casuali (cioè non sequenziali).
- L'utilizzo di più di 12 dischi per vdev è sconsigliato. Il numero ottimale di unità di storage per vdev è compreso tra 2 e 9 (vedere la sezione "5 Prestazioni nel funzionamento regolare").
- All'interno di un vdev si dovrebbero utilizzare dischi con caratteristiche il più possibile omogenee; la situazione ottimale si ha quando tutte le unità di storage sono della stessa marca e modello (ovviamente affidabili), ma provenienti da lotti di produzione differenti, per ridurre il rischio che un eventuale difetto sporadico nella linea di produzione porti al guasto di più di un'unità del vdev.

NOTA: qualora ZFS venga utilizzato con un controller RAID hardware, è fortemente consigliato impostare quest'ultimo in modalità "dischi singoli", affinché ZFS abbia il pieno controllo di ciascuna unità di storage.

Vediamo ora in dettaglio i vari casi di vdev sopra elencati, considerando anche la capacità di archiviazione di ogni soluzione e il grado di tolleranza ai guasti.

2.1 VDEV a singolo disco

Volendo iniziare con un singolo disco e aggiungerne altri in seguito, l'unica possibilità è un pool composto da più vdev, ciascuno costituito da una singola unità di storage. Tale struttura ricorda vagamente il RAID0, pur non essendolo: in ZFS le scritture vengono distribuite tra i vdev del pool in base al relativo spazio libero disponibile, e non mediante "stripe" uniformi come avviene nel classico RAID0. ZFS non implementa il RAID0, in quanto non conforme con il suo obiettivo primario che, come detto, è quello di garantire l'integrità dei dati.

Figura 2: Esempio di pool costituito da 8 vdev, ciascuno composto da un singolo disco

La capacità di storage di questa configurazione è pari alla somma delle capacità delle singole unità di storage, ma qualora uno dei dischi subisca un guasto tutti i dati del pool andranno persi. Tale modalità è quindi fortemente sconsigliata, anche perché non fornisce le prestazioni di uno storage RAID0 (se si ha necessità delle performance di un RAID0 conviene utilizzare Btrfs o altro filesystem sopra MD/RAID0).

Vediamo ora un esempio per creare un pool di due vdev, costituiti rispettivamente dai dispositivi /dev/sda e /dev/sdb. Basta digitare:

```
zpool create tank /dev/sda /dev/sdb
```

ove "tank" è il nome del pool (un qualsiasi termine composto da una lettera seguita dai caratteri "a..z", "A..Z", "0..9", "_", "-", "." e che non inizi con le seguenti parole riservate: c0, .., c9, mirror, raidz, draid, spare e log). Per la descrizione completa del comando "zpool create" vedere la pagina man *zpool-create(8)*. Nella creazione del pool occorre specificare alcune opzioni, ad esempio:

```
zpool create -f -m /mountpoint -o ashift=12 tank /dev/sda /dev/sdb
```

L'opzione "-f" consente di forzare l'uso di dispositivi già utilizzati in precedenza per altri filesystem o altre condizioni anomale (vedere la sopracitata pagina man), l'opzione "-m" permette di scegliere il punto di mount del pool che altrimenti sarebbe "/tank" (il mount in ZFS è automatico, non occorre inserirlo in /etc/fstab), mentre l'opzione "-o ashift=12" è imperativa per gli HDD che hanno settori fisici da 4KB[4]. Il parametro "ashift" è l'unico che non può essere modificato dopo la creazione del vdev ed è estremamente importante poiché un vdev con ashift minore del necessario riduce drasticamente le performance del pool. Per tale motivo, anche se si stanno usando degli HDD con settori fisici da 512 byte (ashift=9) conviene comunque impostare ashift=12 (un giorno si vorrà sicuramente espandere il pool sostituendo quei dischi con modelli recenti di dimensioni maggiori, che avranno sicuramente settori da 4KB). Per gli SSD[5] occorre usare sempre ashift=12 o ashift=13[6].

Per distruggere il pool "tank" appena creato basta digitare:

```
zpool destroy tank
```

Notare che con ZFS è sempre preferibile utilizzare l'intero disco piuttosto che le singole partizioni (ZFS creerà automaticamente il partizionamento necessario). Inoltre, per prendere confidenza con ZFS senza fare danni (ma non per un impiego reale) è possibile simulare dei device virtuali tramite opportuni file, creati nel filesystem di sistema; ad es., per creare in bash un pool di tre vdev virtuali di 1GB l'uno, basta digitare da root:

```
for i in {1..3}; do truncate -s 1G /zfs_fakedev${i}.img; done
zpool create -m /mnt/mypool tank /zfs_fakedev1.img /zfs_fakedev2.img /zfs_fakedev3.img
```

2.2 VDEV ridondanti (MIRROR)

La configurazione più semplice è quella costituita da un pool di un unico vdev, formato da N dischi:

[4] La dimensione dei settori fisici (e logici) di un HDD può essere letta in Linux con il comando "fdisk -l" oppure "hdparm -I" oppure "smartctl -a".

[5] Con il termine "SSD" intendiamo una generica unità a stato solido SATA, SAS o PCIe (NVMe), a meno che il tipo non sia specificato esplicitamente.

[6] Tutte le unità di storage a stato solido hanno settori fisici (pagine flash) di almeno 4KB, ma molti SSD SATA espongono settori fisici di 512 byte (vedere https://forum.proxmox.com/threads/samsung-ssds-a-right-ashift-size-for-zfs-pool.71627/); per evitare l'amplificazione delle scritture (e il rapido esaurimento dell'SSD SATA) è necessario configurare l'opzione ashift a 12 o 13. Viceversa, la maggior parte degli NVMe supporta la doppia modalità 512B / 4KB (la prima mantiene la compatibilità con vecchi OS, la seconda consente di ottimizzare i trasferimenti di dati sul bus PCIe).

Figura 3: Esempio di vdev formato da mirror di 8 dischi

In tal caso i dati vengono replicati sugli N dischi. La capacità di storage è data dalla dimensione del disco più piccolo. La tolleranza ai guasti è estremamente elevata essendo tale soluzione in grado di sopportare il guasto di N-1 dischi, ma lo spreco di archiviazione è enorme, per cui questa configurazione viene impiegata solitamente con due o al massimo tre unità di storage.

Per creare un mirror di due dischi /dev/sda e /dev/sdb basta digitare:

```
zpool create tank mirror /dev/sda /dev/sdb
```

Per semplicità e chiarezza abbiamo omesso le opzioni "-f", "-m /mountpoint" e "-o ashift=12" (viste nella precedente sottosezione), che andranno sempre specificate nei casi reali. Notare che il mirror può anche essere creato per passi successivi, cioè considerando un dispositivo per volta:

```
zpool create tank /dev/sda              (crea il pool 'tank' composto da un vdev a singolo disco)
zpool attach tank /dev/sda /dev/sdb     (aggiunge il secondo device al vdev creando il mirroring)
```

Ciò è particolarmente utile nella transizione da un RAID1 classico a MIRROR ZFS: un modo semplice (ma rischioso) di procedere è quello di marcare come "faulty" uno dei due dischi del RAID1, eliminarlo dall'array e riassegnarlo ad un pool ZFS. Il RAID1 resterà in piedi (anche se degradato) e sarà possibile effettuare la copia dei dati sul filesystem ZFS. Al termine della copia basterà eliminare anche il secondo disco dal RAID1 e attaccarlo al vdev dello zpool creando di fatto il MIRROR ZFS. Si raccomanda caldamente di effettuare prima un backup del RAID1 poiché se il disco rimasto si danneggia durante la copia, tutti i dati andranno persi.

Per quanto riguarda le opzioni viste prima, "-f" e "-o ashift=12" (quando necessarie) devono essere specificate in entrambi i comandi, mentre "-m" deve essere specificato solo in "zpool create".

Una configurazione più interessante è quella relativa ad un pool costituito da più vdev, ove ciascun vdev è un mirror di due dischi:

Figura 4: Esempio di pool costituito da 4 vdev, ciascuno composto da mirror di 2 dischi

I tal caso la capacità di storage è la somma delle capacità dei singoli mirror. L'efficienza di storage (definita come rapporto tra lo spazio di archiviazione disponibile e quello fisico) è del 50% se tutti i dischi hanno uguale capacità, mentre la tolleranza ai guasti varia da 1 a N/2 a seconda di quali dischi si guastano (vedere la sezione "4 Considerazioni sullo stato degradato").

Il pool può essere creato con un unico comando. Consideriamo ad esempio un pool costituito da due vdev i cui device sono /dev/sda + /dev/sdb (primo mirror) e /dev/sdc + /dev/sdd (secondo mirror):

```
zpool create tank mirror /dev/sda /dev/sdb mirror /dev/sdc /dev/sdd
```

In pratica basta utilizzare la parola chiave “mirror” seguita dai dispositivi di archiviazione che costituiscono il mirror, ripetendo tale parola chiave per ogni mirror che si vuole aggiungere. In alternativa è possibile creare il primo mirror (cioè il primo vdev) ed in seguito aggiungere il secondo al pool:

```
zpool create tank mirror /dev/sda /dev/sdb
zpool add tank mirror /dev/sdc /dev/sdd
```

oppure creare un pool con due vdev semplici ai quali attaccare poi le altre unità di storage:

```
zpool create tank /dev/sda /dev/sdc
zpool attach tank /dev/sda /dev/sdb
zpool attach tank /dev/sdc /dev/sdd
```

Notare che la struttura del pool ottenuta assomiglia a quella del classico RAID10, ma non lo è in senso stretto: in un RAID10 classico, costituito ad esempio da 2 unità da 1TB (primo RAID1) più 2 unità da 2TB (secondo RAID1), il secondo TB sulle unità di maggiore capacità verrebbe sprecato e lo spazio di archiviazione risultante sarebbe di 2 TB; mettendo invece le stesse unità in uno zpool di due MIRROR, si ottiene un MIRROR da 1 TB più un MIRROR da 2 TB con uno spazio totale di archiviazione pari a 3 TB. Come detto, le scritture vengono distribuite da ZFS tra i due MIRROR in base alla capacità di storage dei singoli vdev.

Non è semplice caratterizzare le prestazioni di un pool ZFS: quando viene scritto un solo record (vedere il parametro “recordsize” nella sezione “7.4 Dimensione massima dei blocchi di dati”) questo viene allocato su un singolo vdev, mentre quando vengono scritti più record, ZFS li distribuisce “bilanciandoli” tra i vari vdev del pool, quindi le prestazioni in teoria dovrebbero scalare linearmente con il numero di vdev. In pratica esse sono condizionate dal vdev con prestazioni inferiori: in un pool di ‘N’ vdev, le prestazioni saranno pressoché equivalenti a quelle del vdev più lento, moltiplicato per ‘N’. Per cui è fortemente consigliabile utilizzare vdev con stessa struttura e dischi con caratteristiche omogenee (possibilmente unità di stessa marca e modello, ma provenienti da lotti di produzione diversi per scongiurare eventuali difetti di produzione in più unità).

2.3 VDEV con parità singola (RAIDZ1)

La configurazione di base è quella mostrata nella figura seguente:

Figura 5: Esempio di vdev formato da 8 dischi in RAID con parità singola

Ipotizzando che tutti i dischi abbiano uguali caratteristiche, l’efficienza di storage è pari a (N-1)/N, in quanto lo spazio di un disco viene dedicato ai dati di parità. Ad esempio per otto dischi tale efficienza vale 0.875 (87.5%). Questa configurazione è in grado di tollerare il guasto di un solo disco. Notare che i dati di parità sono qui mostrati concentrati in un sola unità di storage, ma in realtà essi vengono distribuiti tra i diversi dischi del vdev[7] (così come i dati veri e propri, che vengono anch’essi segmentati e distribuiti).

Quando si ha a disposizione un numero sufficiente di dischi, nulla vieta di creare un pool costituito da più vdev RAIDZ1: ad esempio, 12 unità di storage si possono ripartire in due vdev RAIDZ1 di 6 dischi ciascuno, oppure tre vdev di 4 dischi. In tal modo le scritture verranno bilanciate sui vdev del pool, incrementando le prestazioni del filesystem, analogamente a quanto visto per pool costituiti da due o più MIRROR.

Per quanto riguarda la creazione di un pool con vdev RAIDZ1, essa è identica alla creazione di un pool con vdev MIRROR, tranne per il fatto che viene utilizzata la parola chiave “raidz” o “raidz1”. L’esempio seguente mostra come creare un pool con un singolo vdev RAIDZ1 composto da tre dischi:

[7] Vedere la pagina man *zpoolconcepts(7)*.

```
zpool create tank raidz1 /dev/sda /dev/sdb /dev/sdc
```

NOTA: un vdev con parità non può essere rimodellato dopo la creazione; in particolare, non si possono aggiungere o togliere dischi, ne convertire il tipo di parità da singola a doppia a tripla (o viceversa).

2.4 VDEV con parità doppia e tripla (RAIDZ2 e RAIDZ3)

Le configurazioni RAIDZ2 e RAIDZ3 sono mostrate nelle due figure seguenti:

Figura 6: Esempio di vdev formato da 8 dischi in RAID con parità doppia

Figura 7: Esempio di vdev formato da 8 dischi in RAID con parità tripla

Ipotizzando che tutti i dischi abbiano uguali caratteristiche, l'efficienza di storage è pari a (N-2)/N per il RAIDZ2 (75.0% nel caso di 8 dischi) e (N-3)/N per il RAIDZ3 (62.5% sempre con 8 dischi), mentre la tolleranza ai guasti è rispettivamente di due e tre unità di storage.

Valgono le stesse considerazioni fatte per il caso di vdev con parità singola. Anche il comando per la creazione del pool è analogo, basta utilizzare le parole chiave "raidz2" e "raidz3".

3 Zpool multiforme

Sebbene ZFS di default rifiuti vdev disomogenei, è possibile comporre un pool con vdev di diversa natura, specificando l'opzione "-f". Tenere sempre a mente che se un singolo vdev fallisce, l'intero pool fallisce con esso: in ZFS non c'è tolleranza ai guasti a livello di pool, ma solo a livello di vdev! Perciò, se un vdev è costituito da un singolo disco, un guasto su quell'unità causerà la perdita irrimediabile dell'intero pool.

Per quanto riguarda le prestazioni, nella sottosezione "2.2 VDEV ridondanti (MIRROR)" abbiamo accennato al fatto che che esse scalano linearmente con il numero di vdev, considerando come fattore base le performance del vdev con prestazioni inferiori. Dunque non ha senso, ad esempio, impiegare un pool costituito da più vdev di cui alcuni sono basati su SSD e altri basati su dischi magnetici (fa eccezione l'impiego di dispositivi a stato solido per lo "special device", come descritto nella sezione "19 ZFS Metadata: special device").

4 Considerazioni sullo stato degradato

Potrebbe essere allettante optare per la configurazione che possiede l'efficienza di storage più alta (RAIDZ1), ma tale scelta è abbastanza rischiosa. Se un disco si guasta, le prestazioni del pool saranno drasticamente ridotte durante la sostituzione e il resilvering. Inoltre in tale fase (che potrebbe durare ore, giorni o settimane, a seconda delle prestazioni, della capacità e del numero dei dischi) il pool non ha alcuna protezione da ulteriori guasti, fino a quando l'array non è stato completamente ricostruito. Pertanto utilizzare RAIDZ1 è un po'

come giocare alla roulette russa (il RAID5 convenzionale è fortemente deprecato esattamente per lo stesso motivo). In pratica, c'è solo un caso d'uso che giustifica il RAIDZ1: quando si hanno solo tre dischi e si desidera massimizzare lo spazio disponibile (ovviamente mantenendo la ridondanza); infatti l'unica alternativa è un mirror a 3 dischi (oppure a 2 con un'unità in spare). Notare che con quattro dischi conviene avere un pool basato su due vdev mirror, mentre con cinque dischi inizia ad avere senso anche la configurazione RAIDZ2.

Come descritto in precedenza, RAIDZ2 e RAIDZ3 forniscono una implementazione più sicura in uno scenario potenzialmente catastrofico, grazie all'impiego rispettivamente di doppia e tripla parità, che consente di tollerare la perdita di un ulteriore disco oppure di due dischi durante il resilvering. Il problema è però solo mitigato, perché le prestazioni di pool RAIDZ2 / RAIDZ3 degradati e il tempo di resilvering sono ben peggiori di quelli del RAIDZ1, essendo i calcoli di parità considerevolmente più complicati, con un impatto tanto più marcato quanto più elevato è il numero di dischi nel vdev e quanto maggiore è il livello di parità.

Per alcuni esperti ZFS la soluzione ottimale è nell'utilizzo dei soli MIRROR. Dal punto di vista dell'efficienza di storage non c'è molta differenza tra il 62.5% del RAIDZ3 (con otto dischi) ed il 50% del MIRROR. Ma quando un disco si guasta in un MIRROR vdev, il pool subisce un impatto minimo: non si devono ricostruire i dati dalla parità, si ha solo un dispositivo in meno da cui effettuare le letture. Anche nel resilvering il pool subisce un impatto minimo: si eseguono semplici letture dal membro rimanente del vdev e semplici scritture sul nuovo membro. Nelle modalità RAIDZ1 / RAIDZ2 / RAIDZ3 sono invece necessarie letture da tutti i dispositivi e complessi calcoli, fino al completamento del resilvering. Per cui il resilvering di MIRROR è molto più veloce di quello di vdev RAIDZ1, RAIDZ2 e RAIDZ3; inoltre esso causa un impatto minore sulle prestazioni del pool e sulle unità di storage. Notare che in un pool di MIRROR la resilienza a più guasti è comunque abbastanza forte; ad esempio, con 8 dischi di stessa marca e modello (4 mirror di 2 dischi ciascuno) si ha che:

- la probabilità di sopravvivere a un guasto di un singolo disco è del 100% (tutti i dischi hanno un partner ridondante);
- la probabilità di sopravvivere al guasto di un secondo disco è del 85.7% (di sette dischi rimanenti, solo uno di essi non ha un partner ridondante, quindi la probabilità che il pool muoia è 1/7, cioè il 14.3%, da cui segue la probabilità di sopravvivenza del 85.7%);
- la probabilità di sopravvivere al guasto di un terzo disco è del 66.7% (di sei dischi rimanenti, solo due di essi non hanno un partner ridondante, quindi la probabilità che il pool muoia è 2/6, cioè il 33.3%);
- e così via, la probabilità (in %) di sopravvivere al guasto di un generico disco è $(1 - f/(N - f)) \cdot 100$, ove 'f' è il numero di dischi già guasti e 'N' è il numero di dischi totali del pool.

Le considerazioni effettuate sopra presuppongono tutti MIRROR a due dischi: con tre dischi per vdev il pool sarebbe ancora più resiliente (ma ovviamente con minore efficienza di storage).

Una domanda sorge spontanea: perché dovremmo scambiare la garanzia di resistenza a due guasti del RAIDZ2 (del 100%) con la probabilità del 85.7% di sopravvivenza nel caso di pool di MIRROR? A causa del tempo molto minore richiesto dal resilvering, e per l'impatto drasticamente inferiore sul pool durante la ricostruzione. Come detto, la ricostruzione di un vdev ridondante è molto meno stressante per le unità di storage della ricostruzione di un vdev con parità, ed è anche molto più veloce.

Indipendentemente dalla struttura finale del pool ZFS, è comunque sempre auspicabile effettuare dei backup regolari del pool (e depositarli altrove). MIRROR e RAIDZ non mettono al riparo da eventi catastrofici quali fulmini, terremoti, inondazioni, atti vandalici, hacking, virus, ransomware, ...

5 Prestazioni nel funzionamento regolare

Matthew Ahrens, uno dei principali sviluppatori di ZFS, ha suggerito che per ottenere le migliori prestazioni su operazioni di I/O casuali occorre utilizzare vdev RAIDZ1 da 3 dischi, RAIDZ2 da 6 e RAIDZ3 da 9, in modo da

riservare sempre 1/3 dello spazio di archiviazione totale alla parità. E che per avere prestazioni ancora migliori, si deve seriamente prendere in considerazione l'utilizzo del mirroring! Non stiamo scherzando: proprio come il RAID10 è stato a lungo riconosciuto come la topologia RAID convenzionale con le migliori prestazioni, un pool di vdev MIRROR è di gran lunga la topologia ZFS con le prestazioni migliori. In sostanza:

- non essere avidi: l'efficienza di archiviazione del 50% del MIRROR è più che accettabile (e non lontana dal 66.6% del RAIDZ1 da 3 dischi o del RAIDZ2 da 6 o del RAIDZ3 da 9);
- a parità di quantità di dischi, un pool di vdev MIRROR supererà notevolmente le prestazioni di un pool di vdev RAIDZ;
- un pool di vdev MIRROR degradato avrà prestazioni migliori di un pool di vdev RAIDZ degradato e si ricostruirà molto più velocemente dopo la sostituzione dell'unità danneggiata;
- un pool di vdev MIRROR è più facile da gestire, mantenere, e aggiornare rispetto a un pool di vdev RAIDZ (vedere la sezione "16 Espansione di vdev ridondanti e con parità").

In conclusione è fortemente raccomandato l'impiego di vdev MIRROR, a meno di avere delle solide ragioni per una scelta diversa, e la completa consapevolezza delle implicazioni future di tale scelta.

6 Comandi offline, online, detach, replace, remove

Se un device di un vdev non funziona correttamente (o si vogliono effettuare dei test), si può indicare a ZFS di ignorare tale dispositivo rendendolo "offline":

```
zpool offline <pool> <device>
```

È possibile porre il device nello stato offline solo se l'operazione non causa perdita di dati (cioè se il dispositivo fa parte di un vdev ridondato o con parità).

Quando un dispositivo è offline ZFS non gli invia alcuna richiesta. Inoltre esso resta in stato offline anche dopo un riavvio del sistema, a meno che non venga usata l'opzione "-t" nel comando precedente, nel qual caso il device viene riportato automaticamente online dopo il riavvio. Per riportare manualmente online un dispositivo basta ripetere il comando precedente con la keyword "online" al posto di "offline".

Un dispositivo offline fa ancora parte del vdev: per utilizzarlo in un altro vdev (o in altro pool o rimuoverlo del tutto) occorre disconnetterlo esplicitamente tramite i comandi "detach" o "replace":

```
zpool detach <pool> <device>
zpool replace <pool> <device> <new_device>
```

Il detach è permesso solo su dispositivi appartenenti a vdev ridondati (MIRROR), mentre il replace è permesso su qualsiasi device, anche su vdev a singolo disco non danneggiati (in tal caso, viene eseguita la migrazione dei dati al nuovo dispositivo e, al termine, il vecchio disco viene rimosso dalla configurazione).

Infine (con OpenZFS $\geq$ 0.8) è possibile rimuovere vdev a singolo disco o interi vdev ridondati (MIRROR) con il comando "zpool remove" :

```
zpool remove <pool> <vdev>
```

ZFS tenterà di spostare gli eventuali dati presenti nel vdev che si desidera rimuovere sugli altri vdev del pool (tale operazione richiede del tempo e dovrà essere pertanto monitorata con il comando "*zpool status*").

NOTA: Attualmente (OpenZFS $\leq$ 2.3.x) l'operazione "remove" non è permessa su pool che contengono vdev con parità (RAIDZ), qualsiasi siano gli eventuali altri vdev che costituiscono il pool. Quindi, se per errore si aggiunge un singolo disco ad un pool contenente un vdev RAIDZ, poi non sarà più possibile rimuoverlo!

Motivo in più per non utilizzare vdev con parità! Questa limitazione non sussiste nella versione ZFS di Oracle, per cui è auspicabile che in futuro tale rimozione venga implementata anche in OpenZFS.

7 Tuning del filesystem ZFS

Analizziamo ora alcune impostazioni finalizzate ad ottimizzare le prestazioni dell'intero filesystem, nonché personalizzare i dataset in base alle loro specifiche destinazioni d'uso. Esistono due tipologie di proprietà:

```
zpool get all <pool>
```

Mostra le proprietà che riguardano l'intero pool ZFS nel suo complesso (ad esempio la capacità complessiva, lo stato di salute, il livello di frammentazione dello spazio libero, ...). Il parametro "ashift" viene mostrato da questo comando. Tali proprietà sono descritte nella pagina man *zpoolprops(7)*; alcune sono read-only. L'ultima colonna mostrata dal comando "zpool get" illustra se una proprietà è stata modificata ("local"), o meno ("default").

```
zfs get all <pool>
zfs get all <pool>/<dataset>
```

Mostra le proprietà relative agli oggetti ZFS, cioè tutto ciò che è configurabile a livello di dataset, volumi e snapshot (ad esempio mountpoint, compressione, encryption, quote,...). Tali proprietà sono descritte nella pagina man *zfsprops(7)*. L'ultima colonna mostrata dal comando "zfs get" illustra se una proprietà è stata assegnata esplicitamente ("local"), se viene ereditata dall'oggetto genitore ("inherited from") o se è quella di default ("default").

Quando una proprietà viene assegnata nel filesystem radice (o in uno specifico dataset), la nuova impostazione si applica a quell'entità e a tutti i suoi figli (anche esistenti) che ereditano la proprietà in questione (cioè quando viene riportato "inherited from" per tale proprietà). Al contrario, i figli per i quali la proprietà è stata impostata esplicitamente ("local") non recepiscono automaticamente le modifiche effettuate sul genitore.

Nei comandi seguenti, il termine <pool> indicherà il generico pool creato e <dataset> un generico dataset contenuto nel pool. Per evitare errori, si raccomanda di configurare inizialmente le impostazioni desiderate per l'intero pool e successivamente personalizzare i dataset in base alle esigenze specifiche.

7.1 Disabilitare l'access time (atime)

A meno di scenari altamente specifici, non è necessario tracciare ora e data di accesso ai file. La disabilitazione di "atime" per l'intero pool riduce notevolmente le scritture sul filesystem migliorando le prestazioni di I/O (inoltre, qualora il pool sia basato su SSD, ciò aumenterà la durata delle relative memorie flash):

```
zfs set atime=off <pool>
```

Per verificare che atime sia stato disabilitato digitare:

```
zfs get all <pool> | grep -w atime      (deve mostrare 'off')
mount | grep <pool>                     (deve mostrare 'noatime' nel mount)
```

7.2 Modificare lo storage degli attributi estesi (xattr)

In OpenZFS < 2.3.0 (su Linux e FreeBSD) gli attributi estesi (in gergo "xattr" o "NFSv4 ACL") vengono memorizzati di default in sottodirectory nascoste, causando molteplici accessi al filesystem. È raccomandato memorizzare invece tali attributi negli inode del filesystem, per generare minori richieste di I/O:

```
zfs set xattr=sa <pool>                (default in OpenZFS ≥ 2.3.0, vedere nota successiva)
```

Per verificare che xattr sia stato correttamente impostato digitare:

```
zfs get all <pool> | grep xattr        (deve mostrare 'sa' solo in OpenZFS < 2.3.0, vedere nota successiva)
```

NOTA: In OpenZFS ≥ 2.3.0 l'impostazione "xattr=on" (attiva di default) è stata resa equivalente a "xattr=sa", mentre il vecchio metodo basato su sottodirectory nascoste è stato etichettato con 'dir'. Quindi a partire da OpenZFS 2.3.0 il comando "zfs get" restituirà 'on' (anziché 'sa') quando gli attributi estesi sono memorizzati negli inode e 'dir' quando vengono memorizzati in sottodirectory nascoste. Vedere la pagina man *zfsprops(7)*.

7.3 Compressione LZ4

La compressione LZ4 permette di diminuire la latenza e il carico di I/O; se un file è incomprimibile ZFS se ne accorge abbastanza presto, trascrivendo tale file inalterato. Con LZ4 il carico sulla CPU è minimo[8], per cui non ci sono controindicazioni ad abilitare tale compressione di default sull'intero pool:

```
zfs set compression=lz4 <pool>
```

Per verificare che la compressione LZ4 sia stata abilitata digitare:

```
zfs get all <pool> | grep compression
```

A livello di pool, si sconsiglia l'uso di altri tipi di compressione, a meno di avere le idee molto chiare sulle motivazioni dietro tale scelta. La modalità di compressione (come molte altre proprietà ZFS) viene ereditata dai dataset del pool e può eventualmente essere modificata: ad esempio, per un dataset che contenga principalmente dati altamente comprimibili si potrebbe forzare "compression=gzip" (in modo da ottenere un rapporto di compressione migliore rispetto a LZ4), ma solo dopo attenta valutazione. I metodi di compressione disponibili sono descritti dettagliatamente nella pagina man *zfsprops(7)*.

7.4 Dimensione massima dei blocchi di dati

In un pool ZFS è possibile stabilire la dimensione massima dei blocchi di dati che vengono allocati dal filesystem (notare che tale dimensione deve essere necessariamente una potenza di 2). Ad esempio, per impostare una dimensione massima di 1 megabyte per tutti i blocchi di dati allocati sull'intero pool, basta digitare:

```
zfs set recordsize=1M <pool>
```

Il valore di default è 128K. Questo è uno di quei parametri che condiziona fortemente le prestazioni del filesystem e va quindi attentamente calibrato all'uso che se ne deve fare. Ad esempio, per un pool di storage multimediale 1M rappresenta il valore ottimale, mentre per lo storage di database è necessario un valore di 16/32K (le motivazioni dietro alla scelta di tali dimensioni esulano dallo scopo di questo documento).

Come già accennato, in ZFS ogni dataset eredita i parametri del pool (o dataset) genitore, a meno che essi non vengano specificatamente riassegnati. È quindi possibile creare, nello stesso pool, più dataset con "recordsize" (e altri parametri) differenti, in modo che ciascun dataset sia ottimizzato per l'uso a cui è destinato. Tale livello di personalizzazione è possibile grazie al fatto che ogni dataset è in pratica un filesystem ZFS diverso (infatti tra i filesystem elencati dal comando "mount" appaiono anche i dataset ZFS).

Come detto sopra, il parametro "recordsize" determina la dimensione massima dei blocchi allocati nel filesystem. Ciò significa che ZFS è in grado di allocare autonomamente anche blocchi più piccoli quando è

[8] Vedere https://klarasystems.com/articles/openzfs1-understanding-transparent-compression/

conveniente (sempre di dimensione pari a una potenza di 2), evitando inutili sprechi di spazio all'interno del pool. Il seguente comando (che va digitato da root o mediante "sudo") permette di vedere alcune statistiche sulle dimensioni dei blocchi allocati nell'intero pool (e altre informazioni):

```
zdb -Lbbbs <pool>
```

7.5 Deduplicazione dei dati

La deduplicazione (deduplication) è una tecnica che elimina le copie di dati ripetuti. È utile qualora più utenti archivino gli stessi dati in posizione diverse o grossi file con piccole differenze (ad esempio Virtual Machine con configurazioni simili). Essa però richiede una quantità enorme di RAM ed ha un impatto notevole sulle prestazioni del filesystem: in pratica si ha un reale guadagno solo qualora i dati siano quasi tutti duplicati almeno 5 volte! Quindi, a meno di specifici scenari, la deduplicazione è assolutamente da evitare. Di default essa dovrebbe essere disabilitata, ma è sempre bene verificare che lo sia:

```
zfs get all <pool> | grep dedup
```

7.6 Copie multiple dei dati

Per specifici dataset di un pool ZFS, è possibile impostare l'archiviazione di due copie dei dati (anche tre, se non si usa la crittografia nativa di ZFS), utilizzando il seguente comando:

```
zfs set copies=2 <pool>/<dataset>
```

Questo è un metodo estremamente povero per fornire a dati critici una maggiore possibilità di sopravvivere ad una eventuale corruzione (*BIT ROT*). Ricordiamo che, di default, ZFS non è in grado di correggere errori di checksum per un blocco di dati presenti su un solo disco, può solo rilevare che è avvenuta una corruzione. Ma se un determinato blocco di dati è stato archiviato due volte sullo stesso disco e una delle copie corrisponde al proprio checksum di convalida, quella copia verrà considerata intatta, ed usata per correggere l'altra copia. Ovviamente tale pratica non fornisce protezione da guasti hardware: "copies=2" non è un sostituto per la ridondanza dei dischi; viceversa, esso diventa inutile su vdev ridondanti (mirror) o con parità (raidz), in quanto il filesystem ha già informazioni sufficienti per riparare i danni di un eventuale BIT ROT.

Notare che in ZFS una corruzione silenziosa è praticamente impossibile, poiché ogni blocco viene sottoposto a controllo tramite checksum (anche con l'impostazione predefinita "copies=1" e senza ridondanza dei dischi). ZFS non restituisce mai dati errati: al massimo restituisce un errore nel tentativo di leggere i dati danneggiati, qualora questi non possano essere recuperati per mancanza di ridondanza, parità o copie multiple.

Tenere sempre a mente che se un dato subisce un alterazione nella memoria principale (quindi prima di essere preso in carico da ZFS), il sistema non ha modo di accorgersene a meno che non sia dotato di memorie ECC (ciò spiega perchè la massima garanzia si ha solo con sistemi dotati di memorie ECC). Tuttavia, una volta che l'informazione è stata scritta nel filesystem, qualsiasi corruzione successiva verrà rilevata da ZFS.

8 Importazione/esportazione e mount del pool al riavvio

Durante la creazione di un pool, ZFS provvede automaticamente a montare il filesystem nel punto di mount specificato (oppure in quello predefinito). Tuttavia, la rilevazione del pool ed il montaggio del filesystem non sopravvivono ad un riavvio del sistema, a meno che non si abilitino alcuni servizi ZFS. Nelle distribuzioni Linux più user-friendly tale abilitazione avviene automaticamente in fase di installazione dei pacchetti ZFS, mentre in altre occorre procedere manualmente; con "systemd" sono necessari almeno i seguenti[9]:

[9] Vedere https://wiki.archlinux.org/title/ZFS

```
systemctl enable zfs-import-cache
systemctl enable zfs-import.target
systemctl enable zfs-mount
systemctl enable zfs.target
```

mentre nel caso di OpenRC[10]:

```
rc-update add zfs-import boot
rc-update add zfs-mount boot
```

In alcune distro è necessario anche il servizio systemd "zfs-load-module" (in ogni caso si consiglia di consultare la documentazione della distribuzione utilizzata). Inoltre, esistono dei servizi ZFS opzionali, tra cui:

- Condivisione automatica del filesystem via NFS o SMB (zfs-share);
- Notifiche via email di cambiamenti di stato o eventi di ZFS (zfs-zed);
- Trim periodico delle unità a stato solido (vedere sezione "20 ZFS Trim");
- Scrub periodico (vedere sezione "21 ZFS Scrub");
- Servizi legati ai volumi.

Per l'attivazione e la configurazione di tali servizi aggiuntivi si rimanda alla documentazione ufficiale di ZFS e alle sezioni indicate.

Un pool può anche essere rilevato e montato manualmente utilizzando il comando "zpool import":

```
zpool import                (lista i pool disponibili)
zpool import <pool>         (importa il pool specificato)
```

Per esportare (smontare) il pool occorre il comando:

```
zpool export <pool>
```

Un pool creato o importato nel sistema, viene aggiunto al file "/etc/zfs/zpool.cache". Tale file memorizza alcune informazioni sul pool (come i nomi dei dispositivi). Se questo file esiste durante l'esecuzione del comando "zpool import", esso verrà utilizzato per determinare l'elenco dei pool disponibili per l'importazione. Se viceversa tale file non esiste, il comando "zpool import" non mostrerà nulla. Se "/etc/zfs/zpool.cache" esiste ma non contiene uno specifico pool, nulla verrà mostrato. In sostanza non si può fare affidamento sul file di cache (di fatto, tale file è prossimo ad essere deprecato, in favore di un servizio di scansione[11]). Il modo migliore per conoscere i pool disponibili per una eventuale importazione è tramite i comandi:

```
zpool import -d /dev/disk/by-id
zpool import -d /dev/disk/by-path
zpool import -d /dev/disk/by-partuuid
```

NOTA: Per zpool composti da vdev basati su file, basta specificare con l'opzione "-d" la directory in cui i vdev risiedono.

Il vantaggio dei tre comandi "import" sopra riportati è molteplice: non solo essi determinano, tra tutti quelli disponibili, i device di storage che fanno parte di pool ZFS (mostrando la struttura di tali pool), ma visualizzano anche le unità di storage tramite un loro identificativo univoco (by-id) o la loro posizione sul bus SATA o SCSI (by-path), o l'ID GPT della partizione utilizzata da ZFS (by-partuuid), piuttosto che tramite il classico nome di dispositivo (sda, sdb, ...) o partizione (sda1, sdb2, ...). Sono circa 10 anni che, in Linux, i nomi dei dispositivi non sono più persistenti, ma vengono assegnati dinamicamente al boot, in base ad un ordine che può cambiare tra riavvii successivi; occorrono opportune regole in "*/etc/udev/rules.d/*" per forzarne la persistenza ed è una

[10] Vedere https://wiki.gentoo.org/wiki/ZFS
[11] Vedere https://github.com/openzfs/zfs/issues/1035

pratica sconsigliata. È quindi molto utile la possibilità di identificare inequivocabilmente i device attraverso altri meccanismi. L'uso dei nomi classici per i dispositivi è comodo in fase di creazione del pool, ma poi conviene esportare e reimportare il pool utilizzando il loro identificativo univoco o il path dei device o il GPT ID: in tal modo, tramite il comando "zpool status", sarà molto semplice risalire alla posizione fisica di una eventuale unità danneggiata. Per individuare una unità di storage è utile anche il seguente comando:

```
lsblk -S --output NAME,HCTL,TYPE,VENDOR,MODEL,REV,SERIAL,TRAN,SIZE | sort
```

che permette di mettere in relazione nome del dispositivo, modello, posizione sul bus e S/N. Prima di sostituire un disco, verificare sempre prima il SERIALE (eventualmente rimuovendo temporaneamente l'unità dal suo alloggiamento fisico se il S/N non è direttamente visibile, ovviamente a macchina spenta e disconnessa dalla rete di alimentazione). In tal modo si eviteranno errori e danni involontari al pool.

NOTA: per poter accedere allo stesso pool ZFS da più distribuzioni Linux diverse (ovviamente in momenti diversi) occorre condividere il file "/etc/hostid" tra le distro interessate, altrimenti si verificheranno errori nell'importazione del pool. Caso tipico è quello in cui al boot si hanno più distribuzioni Linux avviabili.

Se si intende utilizzare lo stesso pool ZFS su host con "hostid" differenti, come nel caso di uno storage removibile, è indispensabile esportare il pool da un host prima di importarlo sull'altro. In caso contrario, il pool risulterà ancora assegnato all'host precedente e l'importazione sul nuovo host non andrà a buon fine.

9 Ripristino del pool in caso di distruzione accidentale

Come visto in precedenza, è molto facile eliminare un pool con il comando "zpool destroy <pool>" (con la conseguente perdita dei dati memorizzati). Qualora ciò accadesse per errore o distrazione, è possibile recuperare il pool se si agisce immediatamente (cioè prima che i dispositivi che appartenevano al pool vengano alterati). Basta digitare:

```
zpool import -D -d <device1> -d <device2> ... <pool>
```

ove <device1>, <device2>, ..., sono i dispositivi che componevano il pool che è andato distrutto. Qualora si tratti di dispositivi virtuali basati su file e non di dispositivi fisici allora l'opzione "-d" può semplicemente specificare la directory (o le directory) contenenti tali file. Se si omette il nome del pool allora il comando precedente elencherà soltanto i pool disponibili e non ancora importati (compresi quelli eliminati, a meno che non si rimuova anche l'opzione "-D"). Si noti infine che, qualora uno dei dispositivi sia danneggiato o mancante (ma facente parte in precedenza di un vdev con ridondanza o parità), è possibile forzare la ricostruzione del pool in uno stato degradato utilizzando l'opzione "-f".

10 Rinominazione del pool

Non esiste un comando per rinominare direttamente un pool, ma è possibile farlo semplicemente attraverso l'esportazione e successiva re-importazione:

```
zpool export <pool>
zpool import <pool_old_name> <pool_new_name>
```

11 Modifica del punto di mount del pool

Per montare il pool in una posizione diversa da quella stabilita in fase di creazione, basta modificare la proprietà "mountpoint":

```
zfs get mountpoint <pool>
zfs set mountpoint=/mnt/<new_mountpoint> <pool>
zfs get mountpoint <pool>
```

È possibile cambiare il mountpoint con il pool montato (non occorre esportarlo e reimportarlo): il path verrà aggiornato immediatamente. Se esistono file aperti sul pool o dati in trasferimento da/verso dataset del pool l'operazione dovrebbe essere rifiutata; comunque è sempre bene verificare che non ci siano applicazioni aperte che stiano usando il pool in quel momento. Se la modifica del mountpoint è permanente, ricordare di aggiornare le configurazioni di eventuali applicazioni che utilizzano percorsi sul pool.

NOTA: Se il pool contiene dei dataset e i percorsi di mount di tali dataset non sono stati modificati esplicitamente, essi erediteranno automaticamente la nuova radice.

12 Ripristino del valore di default di una proprietà

Il comando "*zfs inherit*" permette di azzerare una proprietà modificata, rendendola di nuovo ereditabile da un antenato, o ripristinandola al valore predefinito se nessun antenato ha impostato tale proprietà.

Ad esempio, per ripristinare il punto di mount di un determinato dataset (e dei sui figli) a quello di default stabilito dal mountpoint del pool basta digitare:

```
zfs inherit -r mountpoint <pool>/<dataset>
```

Consultare la pagina man *zfsprops(7)* per un elenco delle proprietà ZFS e i relativi valori predefiniti.

13 Rigenerazione del file di cache di ZFS

Il file "*/etc/zfs/zpool.cache* dovrebbe essere rigenerato automaticamente quando si modifica la configurazione del pool. Qualora, ciò non avvenga (o tale file si corrompa per qualche motivo), si può forzare la sua rigenerazione con il seguente comando:

```
zpool set cachefile=/etc/zfs/zpool.cache <pool>
```

In versioni molto recenti di OpenZFS, il file di cache può anche essere disabilitato, qualora si preferisca l'importazione mediante servizio di scansione (vedere i servizi systemd di ZFS):

```
zpool set cachefile=none <pool>
```

14 Creazione di volumi

Come accennato nella sezione "2 Zpool, vdev e dataset", è possibile riservare spazio nel pool di archiviazione per creare device a blocchi (denominati "volumi" in gergo ZFS). Un volume appare come device nella directory "/dev/zvol/". Ad esempio il seguente comando creerà il volume "vol1" di 2GB nel pool "tank":

```
zfs create -V 2GB tank/vol1
```

e sarà possibile accedere ad esso con il path "/dev/zvol/tank/vol1". Un volume, per default, viene preallocato (la preallocazione è necessaria per prevenire errori che possono causare il danneggiamento dei dati del pool se questo esaurisce lo spazio a disposizione). È anche possibile creare volumi "sparsi" (per i quali non viene

inizialmente riservato spazio) ma per quanto detto tale pratica è rischiosa e assolutamente sconsigliata; inoltre l'accesso a volumi "sparsi" è sicuramente meno efficiente di quello a volumi preallocati.

Un volume ZFS gode dei vantaggi dei vdev sottostanti; sfrutta la ridondanza (o la parità), il copy-on-write, lo scrubbing online, la compressione, la deduplicazione dei dati (se abilitata), la cache ZFS (vedere la sezione "18 ARC, L2ARC (cache)"), ... È possibile fare cose molto interessanti con gli zvol[12]!

L'uso più comune di volumi ZFS è come device di swap[13], ma esiste un bug relativo allo swap su ZFS, che causa una condizione di deadlock dell'OS (alla data corrente non ancora risolto[14]). Un altra possibilità d'uso dei volumi ZFS è come block device per macchine virtuali. Infine, sugli zvol si possono creare altri filesystem come EXT4, XFS, ..., che godranno automaticamente di feature quali compressione, encryption, snapshot, ecc.

NOTA: Occorre fare attenzione se si ridimensiona un volume ZFS, poiché ciò può creare corruzione dei dati nel pool (e la corruzione è quasi certa se si ripristina uno snapshot del volume effettuato prima della variazione di dimensione). Per maggiori informazioni vedere la documentazione di OpenZFS[15].

15 Boot e root filesystem su ZFS, brevi considerazioni

FreeBSD ed alcune distribuzioni Linux supportano nativamente l'installazione su ZFS (per FreeBSD è il filesystem di default). Comunque, per poter essere usato come "/boot" e "/", il pool ZFS deve soddisfare alcune condizioni (si rimanda alla documentazione di OpenZFS). Inoltre, ZFS è generalmente più lento di EXT4 o XFS, e quindi è lecito chiedersi se non sia il caso di utilizzarlo solo per "/home" (o meglio per un'area di storage separata) anziché per l'intero OS. Inoltre, usare un volume ZFS come swap può portare al deadlock del sistema operativo (come accennato in precedenza), mentre l'uso di file di swap (su filesystem ZFS) non è possibile[16].

16 Espansione di vdev ridondanti e con parità

Come noto, non è possibile aggiungere dischi in un vdev RAIDZ, né modificare il livello di parità dopo che il vdev è stato creato (vedere la sottosezione "2.3 VDEV con parità singola (RAIDZ1)"). Tuttavia, è possibile espandere la capacità di archiviazione di un vdev con parità, sostituendo progressivamente i dischi esistenti con modelli di capacità superiore. Per prima cosa occorre attivare l'auto espansione dei vdev del pool:

```
zpool set autoexpand=on <pool>
```

Quindi, procedere con la sostituzione graduale dei dischi, cioè inserendo una nuova unità di maggiore capacità alla volta e attendendo il completamento del resilvering prima di proseguire con la successiva. Dopo la sostituzione ed il resilvering dell'ultimo disco, ZFS espanderà automaticamente la capacità complessiva del vdev (e quindi del pool). Come evidenziato nella sezione "4 Considerazioni sullo stato degradato", tale procedura può richiedere molto tempo e comportare notevoli rischi, specie per vdev di tipo RAIDZ1.

Questo metodo funziona anche con i vdev ridondanti, ma per i MIRROR esiste un modo migliore e più sicuro! In particolare, anche nel caso di un vdev MIRROR a due dischi, è possibile mantenere la ridondanza dei dati: basta aggiungere (anziché sostituire) il primo disco di maggiore capacità al MIRROR, attendere il resilvering, quindi procedere alla sostituzione di uno dei due dischi vecchi con la seconda unità di maggiore capacità, e al termine del secondo resilvering effettuare la rimozione del vecchio disco rimasto (di minore capacità). Notare

[12] Vedere https://tadeubento.com/2024/aarons-zfs-guide-zvols/
[13] Vedere https://askubuntu.com/questions/228149/zfs-partition-as-swap
[14] Vedere https://github.com/openzfs/zfs/issues/7734
[15] Vedere https://openzfs.github.io/openzfs-docs/index.html
[16] Vedere https://askubuntu.com/questions/1198903/can-not-use-swap-file-on-zfs-files-with-holes/1198916

che, qualora fosse possibile aggiungere insieme entrambi i dischi di maggiore capacità è possibile effettuare tutto in un unico passaggio, ottenendo quindi il pool espanso praticamente in metà tempo!

NOTA: prima di effettuare la sostituzione delle unità di storage è bene stressare quelle nuove per accertarsi che non abbiano difetti di produzione. Eseguire almeno i due comandi seguenti su ogni nuovo device (tale fase richiederà probabilmente molto tempo ma è un'operazione caldamente raccomandata):

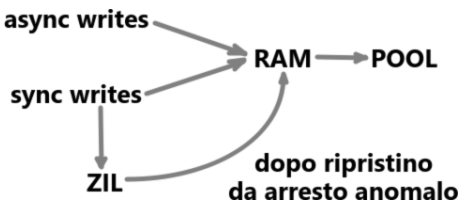
```
dd if=/dev/urandom of=<device> bs=1MiB
smartctl -t long <device>
```

NOTA: L'uso di valori pseudocasuali (*/dev/urandom*) anziché zeri (*/dev/zero*) è preferibile quando si intende utilizzare la cifratura, perché cioè rende difficilmente distinguibili i blocchi cifrati da quelli non utilizzati (usando /dev/zero, lo spazio non ancora scritto apparirebbe come zeri facilmente riconoscibili, rivelando che quei blocchi non contengono dati reali e facilitando eventuali tentativi di attacco alla cifratura del filesystem). Notare che l'uso di */dev/random* (anziché /dev/urandom) è sconsigliato in quanto tale device si blocca quando non c'è abbastanza entropia nel sistema per generare numeri casuali di buona qualità (invece /dev/urandom fornisce comunque valori pseudocasuali, anche se di qualità inferiore). I metodi utilizzabili per aumentare l'entropia di generazione dei numeri casuali in Linux esula dagli scopi del presente testo.

17 Scritture sincrone, ZIL, SLOG (log)

ZFS, come la maggior parte degli altri filesystem, per velocizzare l'accesso ai dati cerca di operare su buffer in memoria invece di accedere direttamente ai dischi: l'OS modifica i dati in memoria e comunica all'applicazione interessata che l'operazione è completata. I dati vengono aggiornati sulle unità di storage solo in un secondo momento, eventualmente aggregando più scritture. Questo è il comportamento predefinito di molti filesystem, incluso ZFS. Tale modalità, detta "*asincrona*", offre buone prestazioni in applicazioni tolleranti ai guasti o in cui la perdita di dati non causa molti danni. Tuttavia, in caso di guasto del sistema o mancanza di alimentazione, tutte le scritture bufferizzate nella memoria principale, non ancora trasferite sul pool, vanno perse.

Lo standard POSIX definisce anche una modalità di scrittura "*sincrona*" in cui, alla notificazione del completamento dell'operazione di scrittura, il filesystem utilizzato deve garantire che una lettura successiva restituirà i dati corretti, indipendentemente da crash del sistema, riavvii, perdita di alimentazione e guasti vari. Pertanto, le applicazioni che desiderano una maggiore garanzia di coerenza possono aprire i file in "modalità sincrona", nella quale le informazioni sono considerate scritte solo dopo essere state realmente trasferite sullo storage. La maggior parte dei database e alcune applicazioni come NFS si basano pesantemente sulle scritture sincrone. Purtroppo, queste sono molto più lente di quelle asincrone[17]. ZFS mantiene una cache di lettura e scrittura molto grande nella RAM di sistema (che può arrivare ad occupare la metà della memoria fisica), ma tale cache è estremamente volatile. Per salvaguardare le scritture sincrone, ZFS utilizza un'area del pool chiamata "ZIL" (ZFS Intent Log) nella quale vengono memorizzati i dati modificati prima che questi vengano distribuiti tra i vari vdev del pool:

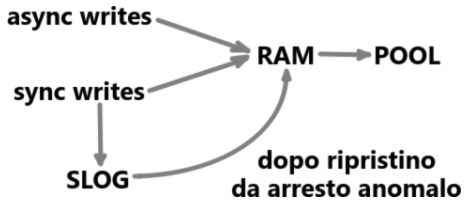


Notare che ZIL non è una cache di scrittura ma un registro di intenti: in condizioni di funzionamento normale, l'aggiornamento del pool viene sempre effettuato a partire dai dati presenti nella cache RAM, mentre quelli memorizzati nello ZIL sono eliminati non appena il gruppo di transazioni associate è stato eseguito con

[17] Alcune unità di storage moderne di classe "enterprise", sono dotate di protezione contro le cadute di alimentazione, che garantisce il salvataggio dei dati nella DRAM interna prima dello spegnimento; il firmware del disco può tenere conto di tale feature per velocizzare le scritture sincrone.

successo sul pool; viceversa, se si verifica un arresto anomalo (prima che il gruppo di transazioni venga trasferito al pool), al mount successivo del filesystem, cioè dopo il reboot, i dati nello ZIL verranno utilizzati per ricostruire le scritture omesse e aggiornare il pool in modo appropriato. Questo è l'unico momento in cui lo ZIL viene letto da ZFS.

Come già anticipato, di default una piccola quantità di spazio del pool viene riservata per agire come ZIL sincrono, quindi le scritture sincrone devono ancora attendere che ZIL venga aggiornato in modo sicuro sui dischi: nel caso di unità di storage magnetiche ciò introduce una latenza significativa, a causa dei movimenti meccanici richiesti ai piatti e alle testine dei dischi. Il vantaggio è che la scrittura sullo ZIL è più veloce della scrittura nella normale struttura del pool, perché non comporta l'aggiornamento dei metadati del filesystem e altre attività di manutenzione. Ma avere lo ZIL sugli stessi dischi del pool introduce una competizione di I/O tra lo ZIL e la normale struttura del pool, con conseguente peggioramento delle prestazioni ogni volta che c'è una porzione significativa di scritture sincrone. Pertanto, nel caso di applicazioni che fanno uso intensivo di scritture sincrone, le prestazioni possono essere migliorate con l'aggiunta di un device SSD di classe enterprise, utilizzando tale dispositivo come "registro di intenti separato" (SLOG):



Gli SSD, avendo minore latenza rispetto alle unità magnetiche, permettono di aggiornare lo SLOG più rapidamente, restituendo all'applicazione la conferma di "scrittura avvenuta" in tempi minori. Ovviamente, anche in questo caso, nell'eventualità di un arresto anomalo, le scritture asincrone presenti in RAM e non ancora trasferite sul pool vanno perse, mentre quelle sincrone salvate nello SLOG vengono recuperate al successivo mount del filesystem (dopo il reboot).

In ZFS la scrittura sincrona/asincrona, può essere forzata per un intero dataset del pool: se in un dataset si imposta il parametro "sync=always" tutte le scritture sul quel dataset vengono gestite come se fossero sempre sincrone, indipendentemente dalla modalità richiesta dall'applicazione:

In tal modo il dataset sarà maggiormente protetto da guasti o perdita di alimentazione (a scapito delle prestazioni che saranno inferiori). Viceversa, se si imposta "sync=disabled", tutte le scritture vengono gestite come se fossero sempre asincrone (le eventuali chiamate alla funzione "sync()" del sistema operativo sono semplicemente ignorate) e quindi nessun dato viene salvato nello ZIL:

In tal caso si avranno le massime prestazioni ma tutte le scritture bufferizzate non ancora trasferite sul pool andranno perse in caso di arresto anomalo del sistema.

Per quanto detto, l'aggiunta di un SLOG ad un pool può velocizzare solo le scritture sincrone (operazioni su database, VM, NFS), mentre è quasi ininfluente sulle prestazioni delle scritture asincrone (anche se si forzano tutte le scritture nello ZIL utilizzando "sync=always"): le scritture asincrone sono sempre aggregate nella cache presente in RAM e restituiscono immediatamente il controllo al chiamante. Pertanto, gli unici miglioramenti diretti riguardano la latenza di scrittura dello SLOG, che essendo su SSD consente alla

chiamata di sincronizzazione di tornare più velocemente. Tuttavia, qualora si faccia un uso intensivo della sincronizzazione, un dispositivo SLOG può accelerare indirettamente anche le scritture asincrone e le letture non memorizzate nella cache: infatti, l'aver trasferito le scritture ZIL su un SLOG separato comporta una minore contesa di I/O sullo storage primario, e quindi un miglioramento delle prestazioni generali del pool.

Un dispositivo SLOG non necessita di una grande capacità di storage, in quanto deve solo accumulare pochi secondi di scritture. Nella maggior parte dei casi una partizione tra 16GB e 64GB è sufficiente (la dimensione massima dovrebbe essere approssimativamente la metà della memoria fisica presente nel sistema).

Ad esempio, per aggiungere un singolo SSD come SLOG del pool, digitare:

```
zpool add -n -o ashift=13 <pool> log <device>   (simulazione)
zpool add -o ashift=13 <pool> log <device>      (aggiunta effettiva)
```

Il dispositivo SLOG viene riportato con il termine "log" nella lista dei componenti del pool (per ottenere l'elenco digitare "zpool status"). Ovviamente, è sempre possibile che i dati nello SLOG vadano persi a causa di guasti nell'SSD; per scongiurare tale evenienza occorre utilizzare due SSD in configurazione MIRROR:

```
zpool add -o ashift=13 <pool> log mirror <device1> <device2>
```

È ovviamente anche possibile utilizzare più SSD in configurazione RAIDZ, ma ciò non ha molto senso poiché diminuirebbe le prestazioni dello SLOG senza apportare alcun beneficio.

NOTA: Per l'SLOG ZFS è imperativo utilizzare SSD di classe enterprise, poiché quelli consumer non sopportano un numero abbastanza elevato di scritture esaurendosi rapidamente, hanno una maggiore latenza, e non integrano un sistema di protezione dalla perdita di alimentazione (batteria tampone o condensatori di elevata capacità) in grado di fornire tensione sufficiente per il tempo necessario a terminare la scrittura sulla flash. Si può ovviare parzialmente a quest'ultima necessità utilizzando un UPS (gruppo di continuità), ma ciò non mette al riparo da guasti all'alimentatore o all'UPS stesso, né alla rapida usura del disco a stato solido. La soluzione richiede necessariamente dispositivi di classe enterprise, che sono purtroppo molto più costosi.

Ricordare sempre che ashift=12 o ashift=13 è mandatorio per gli SSD. Il firmware di molti SSD SATA riporta una dimensione di 512 byte per i settori fisici, ma tutti gli SSD hanno una dimensione di almeno 4KB (alcuni modelli enterprise 8KB, i modelli consumer anche 16KB o 32KB). Un ashift più grande del necessario non comporta grosse penalità, mentre un ashift minore del necessario causa la moltiplicazione delle operazioni di scrittura (con fine prematura dell'SSD e peggioramento delle performance). Per verificare quale valore è stato usato per "ashift" al momento della creazione del pool, basta utilizzare il comando "zpool get ashift <pool>".

NOTA: Ciascun pool ZFS richiede un dispositivo SLOG separato, e non si deve utilizzare lo stesso dispositivo per la cache L2ARC (vedere prossima sezione) altrimenti le prestazioni ne risentiranno. Inoltre, per quanto detto sopra, server multimediali o di backup o sistemi con carichi prettamente asincroni e sequenziali, non beneficiano di un dispositivo SLOG, che viene totalmente ignorato!

18 ARC, L2ARC (cache)

Con ARC (Adaptive Replacement Cache) si intende una cache adattativa di lettura in RAM, per il filesystem ed i volumi. ZFS utilizza sempre questa tipologia di cache. L2ARC fornisce invece un 2° livello di ARC che risiede su un sistema di archiviazione veloce (SSD preferibilmente con interfaccia PCIe). Una cache di secondo livello può sembrare allettante ma non sono tutte rose e fiori: innanzitutto, anche se immagazzinata su SSD, L2ARC è comunque considerata volatile da ZFS (al riavvio del sistema viene rigenerata), utilizza una quantità significativa di RAM per l'indicizzazione (sottratta ad ARC), e anche l'NVMe più veloce è almeno un ordine di grandezza più lento della RAM. La maggior parte dei sistemi subisce una diminuzione di prestazioni dopo che

un dispositivo L2ARC è stato aggiunto. ZFS può beneficiare di L2ARC solo per enormi carichi di lavoro di lettura casuale, distribuiti su centinaia di utenti. In tutti gli altri casi conviene sempre espandere la RAM .

Il dispositivo L2ARC viene riportato con il termine "cache" nella lista dei componenti del pool. Per aggiungere un disco SSD come cache digitare:

```
zpool add -n -o ashift=13 <pool> cache <dev>   (simulazione)
zpool add -o ashift=13 <pool> cache <dev>      (aggiunta effettiva)
```

NOTA: La cache L2ARC non può essere creata come mirror (non ha senso) ma può beneficiare di accessi multipli in lettura ottenuti impiegando più dispositivi indipendenti (quindi dischi, non partizioni):

```
zpool add <pool> cache <dev1> <dev2>
```

Per un uso corretto di L2ARC sono necessari almeno 64GB di RAM (non tentare di utilizzare L2ARC su sistemi con minore quantità di RAM o le prestazioni ne soffriranno terribilmente). Inoltre, la dimensione di L2ARC non deve superare dieci volte la quantità di RAM del sistema.

Dopo che i dispositivi L2ARC sono stati aggiunti, questi vengono riempiti gradualmente con il contenuto della memoria principale (il tempo necessario affinché la cache raggiunga la piena funzionalità varia a seconda delle sue dimensioni). Per monitorare la capacità utilizzata e le operazioni di lettura utilizzare:

```
zpool iostat -v pool 5
```

NOTA: Se si verifica un errore di lettura su un dispositivo L2ARC, la stessa operazione di I/O viene riemessa sul dispositivo originale del pool (bypassando quindi la cache).

Riepilogando, sistemi con carichi principalmente asincroni e sequenziali (come server multimediali o di backup) non beneficiano di L2ARC. L'uso di L2ARC su tali sistemi ha come unico effetto quello di degradare le prestazioni dello zpool! Occorre quindi valutarne attentamente l'impiego.

19 ZFS Metadata: special device

In ogni pool ZFS esiste una "classe speciale" di elementi formata da metadati, blocchi indiretti[18] ed eventuali tabelle di deduplicazione. Questa classe, può essere allocata su uno "special device" costituito da uno o più SSD: ciò permette di accelerare notevolmente l'accesso a tali costrutti del filesystem, poiché essi vengono immagazzinati su dispositivi a stato solido, che hanno bassa latenza e maggiore banda di I/O. Comandi come "ls -lR", "du -h" e simili (e alcune operazioni di manutenzione) diventano molto più veloci. Ma anche l'accesso ai dati veri e propri risulta più spedito, in quanto dati speciali e dati normali viaggiano su canali fisici diversi. È inoltre possibile includere, nello special device, piccoli blocchi di dati o interi dataset, in modo da velocizzare l'accesso anche ad una parte dei dati del pool (ovviamente non tutti i dati possono essere allocati nel device speciale, altrimenti non servirebbe più il pool principale). In sostanza, lo special device permette di ottenere uno storage ibrido con caratteristiche intermedie tra HDD e SDD / NVMe.

NOTA: Quando il pool principale è ridondato, anche lo special device deve esserlo: infatti, se si perdono i metadati, anche i dati veri e propri che risiedono nei altri vdev diventano irraggiungibili. Il tipo di ridondanza dello special device non deve necessariamente coincidere con quella del pool principale, ma è buona pratica che la resilienza ai guasti sia la medesima: si può quindi avere un RAIDZ2 per il pool principale e un MIRROR a 3 dischi per lo special device. È anche possibile aggregare più MIRROR per avere maggiore banda di I/O. L'importante è utilizzare lo stesso tipo di SSD (per bilanciare bene l'I/O) e mantenere la coerenza di "ashift".

Ad esempio, per aggiungere al pool "tank" uno special device composto da 2 vdev MIRROR, realizzati con quattro NVMe (ipotizzando che il pool sia stato creato con ashift=13), basta digitare:

[18] Vedere https://en.wikipedia.org/wiki/Inode_pointer_structure

```
zpool add -o ashift=13 tank special mirror /dev/nvme0n1 /dev/nvme1n1 mirror /dev/nvme2n1 /dev/nvme3n1
```

Si può anche procedere per passi successivi con “zpool add” e “zpool attach”, come visto per i normali vdev.

Quanto spazio è necessario per lo special device? La regola empirica è riservare per i metadati almeno lo 0.3% della dimensione del pool principale; il resto è utilizzabile per piccoli blocchi di dati[19] o dataset particolari (vedere la prossima sezione). ZFS smette di allocare alcuni tipi di oggetti nello special device quando la sua occupazione arriva al 75% (il parametro è regolabile[20]): in tale eventualità, gli oggetti tornano ad essere memorizzati nel pool principale. Ad esempio, per uno storage effettivo di 20TB, e special device su SSD di 960GB si ha: $960 \cdot 0.75 - 20 \cdot 1024 \cdot 0.003 \approx 660$GB che possono essere utilizzati per altro!

19.1 Dati su special device

Per ciascun dataset è possibile forzare ZFS ad allocare sullo special device tutti i blocchi di dati con dimensione minore o uguale ad un valore specificato. Ad esempio con il seguente comando:

```
zfs set special_small_blocks=64K <pool>/<dataset>
```

si impone a ZFS di memorizzare nella classe “special” (anziché nel pool principale) tutti i blocchi di dimensioni minori o uguali a 64KB e appartenenti al dataset indicato. Se la dimensione specificata è uguale alla dimensione massima (“recordsize”) del dataset, allora tutti i blocchi di dati di quel dataset verranno allocati nello special device. In tal modo è possibile spostare interi dataset su SSD in modo da velocizzarne l’accesso.

Per default il parametro “special_small_blocks” ha valore 0, per il pool e per tutti i dataset in esso contenuti (ciò significa che di default nessun blocco di dati viene memorizzato nella classe special).

NOTA: ZFS non sposta mai deliberatamente dati o metadati già immagazzinati. Per spostare i metadati e altri oggetti già esistenti sul device speciale, occorre creare un nuovo dataset, copiare i dati, eliminare il vecchio dataset e quindi rinominare quello nuovo. Questo è anche l’unico modo per alterare alcune proprietà di un dataset, come la dimensione del record, la crittografia e la compressione.

OpenZFS 2.3.4 introduce il nuovo comando “zfs rewrite“[21] che consente di riscrivere il contenuto di specifici file (e cartelle) in una posizione diversa, aggiornando i file in base agli eventuali parametri del dataset che sono stati modificati (ed eventualmente spostando i relativi blocchi sullo special device). Il vantaggio di tale comando è che non passa per lo spazio utente quindi è più veloce di una copia manuale (inoltre opera un file alla volta, permettendo l’aggiornamento anche quando c’è poco spazio sul pool). Il comando “zfs rewrite“ è estremamente recente, quindi non ancora disponibile nella maggior parte delle installazioni Linux; inoltre è per ovvi motivi poco testato, e anche qualora fosse disponibile si consiglia di utilizzarlo con cautela.

20 ZFS Trim

È ben noto che i filesystem su SSD beneficiano dell’esecuzione periodica del comando TRIM, il cui scopo è comunicare alle unità a stato solido quali settori del filesystem non sono più in uso e possono essere cancellati e riutilizzati. Senza il comando TRIM, le unità FLASH non hanno modo di sapere che alcuni settori contengono informazioni non più valide (e non più necessarie) finché il sistema non comunica alle unità di scrivere nuove informazioni in quel punto. Le celle di memoria FLASH devono essere cancellate prima di poter ricevere nuove informazioni e tale operazione è piuttosto lenta; l’uso di TRIM permette di effettuare le cancellazioni in

[19] Vedere https://forum.level1techs.com/t/zfs-metadata-special-device-z/159954/65
[20] Vedere https://forum.proxmox.com/threads/zfs-special-device-75.125956/
[21] Vedere https://github.com/openzfs/zfs/pull/17246

background (quindi prima che le celle FLASH vengano interessate da una nuova scrittura), aumentando le prestazioni generali delle unità a stato solido. Inoltre, TRIM permette anche di bilanciare meglio le scritture sulla FLASH (Wear leveling[22]) in modo da incrementare la longevità delle unità di storage.

Generalmente, in Linux il trimming dei filesystem si effettua con "fstrim -a" (esplicitamente, oppure mediante un opportuno "timer systemd" o tramite "cron"), ma ZFS ha il suo comando dedicato:

```
zpool trim <pool>
```

Anche in questo caso, lo stato del comando può essere esaminato con "zpool status". Il comando TRIM non ha effetto sugli eventuali dischi magnetici (interessa solo le unità a stato solido), quindi in genere l'operazione di trimming viene completata in pochi minuti. È anche possibile attivare l'autotrim, ma fortemente sconsigliato in quanto sottopone le unità di storage a maggior stress ed interferisce con le performance dei dischi: un TRIM periodico settimanale (avviato autonomamente mediante "timer systemd" o "cron") costituisce la migliore soluzione per ottimizzare il funzionamento delle unità a stato solido con ZFS.

Per ulteriori informazioni si rimanda alla pagina man *zpool-trim(8)*.

21 ZFS Scrub

Abbiamo visto precedentemente che una corruzione silenziosa dei dati è impossibile in ZFS: tutto viene sottoposto ad hashing e ogni volta che ZFS legge un blocco di dati (o metadati), confronta quel blocco con la sua checksum e lo corregge automaticamente qualora il blocco (e relativa checksum) siano ridondati, altrimenti si limita a segnalare la corruzione rilevata. Tuttavia nel pool ci possono essere molte informazioni che, dopo essere state scritte, non subiscono letture per molto tempo: su di esse ZFS non esegue normalmente alcun controllo. Esiste la possibilità di forzare esplicitamente un controllo di integrità sull'intero pool:

```
zpool scrub <pool>
```

Lo scrub riparerà automaticamente eventuali blocchi corrotti in ogni vdev con ridondanza, dati di parità o copie multiple, mentre per i vdev funzionanti ma degradati potrà soltanto segnalare i blocchi corrotti.

NOTA: Questo comando, andando a verificare tutto il filesystem, è potenzialmente in grado di rilevare anche eventuali problemi hardware presenti in zone dei dischi poco utilizzate (che in condizioni normali di funzionamento, avrebbero poche probabilità di emergere), pertanto se ne consiglia l'impiego periodico almeno una volta al mese. Notare che lo scrub può richiedere da pochi minuti a diversi giorni, in funzione della struttura del pool, delle sue dimensioni e della tipologia di storage utilizzato. Ecco alcuni esempi reali, relativi a pool da me utilizzati (tutti su unità SATA, connessi alla motherboard o via USB3 10Gbps):

Configurazione ZPOOL	Unità di storage	Connessione	Spazio occupato	Durata scrub	Velocità scrub
SSD partizione (830GB)	Crucial MX500 2TB	SATA	356GB	13m 19s	2.24 s/GB
SSD intero	Samsung 860 EVO 1TB	USB3 10Gbps	485GB	19m 31s	2.41 s/GB
HDD mirror + SSD mirror per special device	2 x WD Ultrastar HC570 22TB + 2 x Kingston DC500M 960GB	SATA	5.22TB	6h 6m	4.11 s/GB
HDD mirror	2 x WD Gold 8TB	USB3 10Gbps	703GB	1h 38m	8.36 s/GB

Come si può osservare, lo scrub è abbastanza lento sui soli dischi magnetici, viene accelerato dalla presenza dello "special device" su SSD SATA (grazie al fatto che almeno i metadati sono localizzati sulle unità a stato solido) ed è più veloce per pool interamente immagazzinati sulle unità FLASH (non ho pool su SSD NVMe al

[22] Vedere https://en.wikipedia.org/wiki/Wear_leveling

momento, ma immagino che sia ancora più veloce). Ovviamente, a parità di hardware, maggiore è la quantità di dati presenti nel pool, maggiore sarà il tempo richiesto per completare la scansione del filesystem.

Lo stato di avanzamento dello scrub (ed eventuali problemi rilevati) possono essere visualizzati con il comando "zpool status". Il filesystem è perfettamente utilizzabile mentre viene sottoposto a scansione; ovviamente, a causa di una componente di I/O destinata allo scrub, le sue performance generali saranno inferiori rispetto al caso d'uso normale, pertanto sarebbe meglio avviare lo scrub in orari o giorni in cui il sistema è sottoposto a minor carico (magari schedulandolo tramite "timer systemd" o "cron"). Comunque, qualora fosse necessario, è possibile sospendere la scansione con il comando:

```
zpool scrub -p <pool>
```

Per continuare dal punto interrotto (anche dopo un reboot o un "export" seguito da "import"), basta utilizzare il primo comando visto sopra. Per ulteriori informazioni si rimanda alla pagina man *zpool-scrub(8)*.

22 Monitorare l'I/O su ZFS

Per visualizzare le richieste di I/O sul pool ZFS (incluso il trim) basta utilizzare questo comando:

```
zpool iostat -r <pool>
```

23 History

È possibile elencare tutte le operazioni che l'utente ed il sistema hanno effettuato sul pool ZFS, tramite il seguente comando (notare che tale elenco storico non è modificabile, né eliminabile):

```
zpool history <pool>
```

Il nome del pool può anche essere omesso, in tal caso verranno elencate le operazioni effettuate su tutti i pool.

24 Metodologia CoW (copy-on-write)

In un sistema operativo, il filesystem gioca un ruolo molto importante. Per Linux sono disponibili vari filesystem, che rientrano quasi tutti in una delle seguenti tre categorie: filesystem di rete (che qui non ci interessano), quelli basati sul Journaling, e quelli basati sul copy-on-write (CoW). ZFS rientra in questa terza categoria. Journaling e CoW sono due tecniche che hanno lo stesso obiettivo: rendere il filesystem più affidabile e preservare le sue strutture interne in caso di crash del sistema, guasti hardware e interruzione improvvisa dell'alimentazione; tuttavia esse assolvono a tale compito in modo molto diverso.

Sulla maggior parte dei filesystem, durante un'operazione di scrittura i dati precedenti (se esistenti) vengono persi in modo irreversibile; in ZFS e negli altri filesystem CoW, invece le modifiche ai dati vengono eseguite su una copia dei dati anziché sui blocchi di dati originali. In altre parole, le modifiche vengono archiviate in una posizione diversa dello storage e i metadati vengono aggiornati per puntare a quella posizione (anche i metadati vengono copiati in scrittura e un'operazione di sincronizzazione scambia il vecchio contenuto con il nuovo). Dopo che i nuovi blocchi di dati e metadati sono stati correttamente validati, i blocchi precedenti vengono contrassegnati come obsoleti (in modo da poter riutilizzare, in seguito, lo spazio che essi occupavano). Questo meccanismo garantisce che, durante la scrittura, i vecchi oggetti siano comunque conservati in caso di interruzione di corrente, crash fatale o guasto hardware, e che quindi al riavvio del sistema sul filesystem siano presenti o le nuove informazioni completamente valide, oppure le vecchie perfettamente integre. Viceversa,

nei filesystem Journaled, in caso di problemi, dati e metadati rischiano sempre di essere lasciati in uno stato inconsistente (che, grazie al Journal, viene corretto nella maggior parte dei casi, ma vi possono essere situazioni che portano a perdite di informazioni).

In sostanza, il vantaggio del CoW è la sua capacità di apportare modifiche "atomiche" al filesystem (cioè il filesystem è sempre perfettamente nello stato precedente oppure nel nuovo stato, ma mai in uno stato intermedio, indipendentemente dall'eventuale tipo di evento catastrofico che si presenta). Ovviamente il CoW comporta un maggior numero di operazioni, per cui un filesystem basato su CoW è generalmente meno veloce di uno basato sul Journaling. Ma l'obiettivo principale dei filesystem CoW non sono le performance quanto piuttosto l'integrità dei dati (grazie anche al checksum esteso ai dati e alla ridondanza). Notare che, in un confronto con gli altri filesystem CoW, ZFS vince su tutta la linea: non solo è decisamente più performante di Btrfs ma ha anche molte caratteristiche in più. Bcachefs invece è ancora immaturo, ed essendo al momento un progetto gestito da una sola persona non è scontato che diventi un prodotto realmente utilizzabile.

In definitiva, con l'hardware attualmente disponibile, ZFS è perfettamente in grado di soddisfare le esigenze di un'ampia categoria di utenti, dai grandi datacenter agli impieghi SOHO (Small Office Home Office), per cui non vi è nessuna ragione valida per non utilizzarlo. È persino disponibile un porting per Windows[23] che, alla data di stesura di questo documento, dovrebbe essere in "stato beta", almeno secondo quanto dichiarato dagli sviluppatori (personalmente non l'ho mai provato e, francamente, non credo che lo farò mai: ipotizzare di poter realizzare un sistema di storage affidabile con Windows significa andare in cerca di guai).

25 Snapshot e cloni

Nella sezione "2 Zpool, vdev e dataset" abbiamo visto che ZFS ha la capacità di creare "snapshot", cioè istantanee non modificabili di particolari dataset (o volumi), create in un momento specifico. Grazie al CoW, gli snapshot non richiedono praticamente nessuno spazio aggiuntivo all'interno del pool. È possibile ripristinare snapshot completi, consentendo così di recuperare i dati presenti nello snapshot.

I "cloni", invece, sono copie modificabili degli snapshot. All'inizio un clone non richiede spazio aggiuntivo nel pool; quando vengono scritti nuovi dati sul clone, vengono allocati nuovi blocchi e le sue dimensioni aumentano; inoltre, la sovrascrittura dei blocchi nel filesystem (o volume) clonato determina la diminuzione del conteggio dei riferimenti sul blocco originale. Infine, poiché un clone è dipendente dallo snapshot utilizzato per crearlo, non è possibile eliminare lo snapshot originale se non si eliminano prima tutti i suoi cloni.

Snapshot e cloni sono argomenti abbastanza avanzati (non tutti ne hanno bisogno) e si possono fare danni con il ripristino degli snapshot, per cui rimando l'approfondimento alla documentazione ufficiale di OpenZFS.

26 Native encryption

Un'altra funzionalità importante di ZFS è la crittografia nativa, introdotta in OpenZFS 0.8. Essa consente di crittografare in modo trasparente i dati all'interno di ZFS stesso, eliminando la necessità di un layer aggiuntivo come LUKS (Linux Unified Key Setup)[24]. Attualmente, ZFS è l'unico filesystem di classe enterprise sufficientemente maturo con strategia CoW, checksum dei dati e crittografia nativa (infatti è la spina dorsale di moltissimi sistemi aziendali, inclusi grandi datacenter e supercomputer). Purtroppo, Bcachefs ha tali caratteristiche ma è ancora in fase molto embrionale, mentre Btrfs ha il CoW ed il checksum dei dati, ma non la crittografia nativa e, cosa peggiore, soffre ancora di molteplici problemi (basta fare una ricerca con "btrfs issues" per rendersene conto). Ci sarà una ragione se Btrfs è stato completamente abbandonato da Red Hat!

La crittografia in ZFS permette di fare cose straordinarie quali:

[23] Vedere https://github.com/openzfsonwindows/openzfs/releases
[24] Vedere https://blog.elcomsoft.com/2021/11/protecting-linux-and-nas-devices-luks-ecryptfs-and-native-zfs-encryption-compared/

- assegnare chiavi di crittografia diverse ai vari dataset (ciò permette ad ogni utente di avere la sua HOME, ed eventuali altre aree, protette con le proprie chiave di crittografia);
- si possono creare snapshot e inviarli/riceverli senza avere la chiave di crittografia; ciò permette di eseguire backup su filesystem / server esterni, senza rivelare la chiave;
- è possibile effettuare lo scrub ed altre operazioni di manutenzione senza aver caricato in precedenza le chiavi di decrittazione: non occorre consegnare a root le chiavi del regno!

Come riportato nella pagina man *zfs-load-key(8)*, ZFS cifra i dati contenuti in file e volumi, gli attributi dei file, le ACL, i bit dei permessi, il contenuto delle directory, la mappatura FUID e le informazioni userused/groupused (contatori di spazio utilizzato per utente e gruppo, vedere pagina man *zfs-userspace(8)*). ZFS non cifra i metadati correlati alla struttura del pool (inclusi i nomi di dataset e snapshot), la gerarchia di dataset, le proprietà, le dimensioni dei file, gli eventuali buchi nei file e le tabelle di deduplicazione (sebbene i dati deduplicati siano comunque cifrati)[25]. Tali "leak" sono considerati insignificanti ai fini di eventuali tentativi di decrittazione non autorizzata. Comunque, se tale protezione non è ritenuta sufficiente l'unica soluzione è lo stacking di ZFS sopra LUKS, ma in tal caso si perdono i vantaggi sopra illustrati (la chiave LUKS va sempre caricata); inoltre, lo scrub con ZFS su LUKS è molto più lento di quello con encryption nativo.

L'algoritmo di default utilizzato per la crittografia ZFS è "*aes-256-gcm*" (in OpenZFS $\geq$ 0.8.4), ma può anche essere specificato esplicitamente tra quelli disponibili (tutti basati su AES). Attualmente, "aes-256-gcm" è considerato il più robusto tra quelli supportati, ma anche quello con il maggiore impatto sulla CPU; nonostante ciò è ben tollerato anche su CPU vecchie di 10 / 12 anni. Le prestazioni sono paragonabili a quelle dei pool non crittografati, indipendentemente dalle dimensioni dei file: da benchmark effettuati, si stima un impatto inferiore al 10% (che è anche minore di quello imposto da LUKS); inoltre, OpenZFS 2.4.0 RC1 ha introdotto il supporto alle istruzioni AVX2, migliorando ulteriormente le prestazioni dei pool crittografati[26].

Due osservazioni importanti, inerenti la crittografia nativa di ZFS, sono le seguenti:

- la deduplicazione può influire sulla sicurezza e quindi dovrebbe essere disattivata per i dati critici (ma abbiamo già visto che è fortemente sconsigliata a meno di scenari specifici);
- la compressione viene applicata prima della crittografia: i dataset potrebbero essere vulnerabili a un attacco di tipo CRIME[27] se "le applicazioni" che accedono ai dati lo consentono; è una possibilità remota (per lo più accademica) ma esiste: per scongiurare del tutto tale possibilità basta disabilitare la compressione nei dataset criptati.

Come la maggior parte delle proprietà ZFS, anche la crittografia può essere abilitata sull'intero pool e/o personalizzata sui singoli dataset. Ricordare che tale crittografia nativa avviene a livello di blocco (come la compressione): quando la si attiva su un dataset esistente, solo i blocchi scritti dopo tale abilitazione vengono crittografati; pertanto è caldamente raccomandato abilitarla fin dal momento della creazione del pool o dei dataset interessati, per non lasciare in giro residui di dati in chiaro.

Una domanda che sorge spontanea è se l'uso della crittografia in ZFS impatta sulla capacità di garantire l'integrità dei dati nel filesystem (cosa che spesso avviene nei filesystem tradizionali). La risposta è no! Un blocco crittografato viene scritto completamente e sopravvive ad un evento catastrofico, oppure viene ripristinato come se non fosse mai stato modificato. In ogni caso non c'è corruzione dei dati[28].

Vediamo dunque come abilitare l'encryption su un dataset e verificare che funzioni:

```
zfs create -o encryption=on -o keyformat=passphrase -o keylocation=prompt <pool>/<dataset>
zfs unload-key <pool>/<dataset>
zfs load-key <pool>/<dataset>
```

[25] Vedere https://www.reddit.com/r/zfs/comments/ienyuz/do_file_sizes_leak_with_zol_encryption/
[26] Vedere https://www.phoronix.com/news/OpenZFS_2.4-rc1-Released
[27] Vedere https://iacr.org/archive/fse2002/23650264/23650264.pdf
[28] Vedere https://www.reddit.com/r/zfs/comments/1ad88wn/zfs_encryption_and_power_loss/

27 IO Scheduler

ZFS ha il proprio scheduler di I/O, pertanto si raccomanda di forzare lo scheduler di I/O del kernel Linux a "none" per tutte le unità di storage completamente dedicate a pool ZFS: ciò eviterà che lo scheduler Linux venga impilato sopra lo scheduler ZFS, degradando le prestazioni del filesystem. Tale accortezza vale particolarmente per le unità di storage magnetiche, ma è consigliata anche per le unità a stato solido.

Il modo più semplice per impostare automaticamente lo scheduler di I/O è tramite una "regola udev". Poiché i nomi dei dispositivi /dev/sdX non sono più persistenti, si deve utilizzare una delle seguenti variabili per identificare le unità di storage dedicate a ZFS:

- DEVPATH → percorso hardware del dispositivo;
- ENV{ID_SERIAL_SHORT} → S/N del dispositivo;
- ENV{ID_PART_TABLE_UUID} → UUID GPT.

Ad esempio, se ci sono 4 dischi che formano il pool ZFS, connessi alle interfacce SATA da 1 a 4, una possibile regola udev è la seguente:

```
# MAP DISK BY PATH (SATA 1 to 4), see "udevadm info /dev/<device>".
ACTION=="add|change", KERNEL=="sd[a-z]", DEVPATH=="*0000:*:00.1/ata1/host*", ATTR\{queue/scheduler\}="none"
ACTION=="add|change", KERNEL=="sd[a-z]", DEVPATH=="*0000:*:00.1/ata2/host*", ATTR\{queue/scheduler\}="none"
ACTION=="add|change", KERNEL=="sd[a-z]", DEVPATH=="*0000:*:00.1/ata3/host*", ATTR\{queue/scheduler\}="none"
ACTION=="add|change", KERNEL=="sd[a-z]", DEVPATH=="*0000:*:00.1/ata4/host*", ATTR\{queue/scheduler\}="none"
```

Qualora solo una partizione sia dedicata a ZFS (ad esempio ciò potrebbe essere vero per un laptop) non è consigliabile forzare lo scheduler di I/O di Linux a "none", in quanto ne risentirebbero negativamente i filesystem presenti nelle altre partizioni dello stesso disco. Pertanto, quando si ha un unica unità di storage destinata sia a ZFS che ad altri filesystem Linux, è bene lasciare attivo lo scheduler di default, accettando un leggero degrado delle performance di ZFS (che su SSD è praticamente impercettibile).

28 ZFS su supporti rimovibili

ZFS non è attualmente resiliente nei confronti di errori di I/O, per cui non è molto adatto ad essere utilizzato su storage removibile (sebbene qualcuno, me compreso, lo usi) poiché falsi contatti sui connettori USB possono provocare il blocco dei processi ZFS all'interno del kernel Linux, costringendo ad un riavvio del sistema tramite il "Magic SysRq Key" e la sequenza R.E.I.S.U.B.[29]) ogni volta che si verifica un falso contatto. Purtroppo resta, al momento, l'unico filesystem CoW con checksum sui dati e encryption nativo; tutte le altre soluzioni di backup sicuro costringono all'uso di più layer, con la scomodità e i rischi che ne conseguono (come detto, Bcachefs è immaturo e Btrfs non ha ancora l'encryption nativo, senza considerare tutti i sui problemi).

Personalmente utilizzo ZFS da diversi anni, anche su dispositivi removibili, e non ho mai perso dati, né ho mai riscontrato una situazione che non fosse recuperabile, ma non posso garantire che un problema grave non possa accadere. Per cui consiglio, specialmente ai meno esperti, di utilizzare ZFS esclusivamente su storage fisso (ovviamente con hardware di buona qualità).

29 Esempi d'uso

Le successive sottosezioni forniscono esempi concreti di come creare ed espandere pool ZFS; fanno riferimento a casi d'uso personali ma possono essere utilizzate come guide di uso generale. Ovviamente, i tipi di unità di storage utilizzate, i nomi dei device e varie altre informazioni riportate di seguito sono riferite alla mia personale configurazione, ma le procedure sono mutuabili senza difficoltà ad altri contesti.

[29] Vedere https://en.wikipedia.org/wiki/Magic_SysRq_key

29.1 Conversione di MD/RAID1 in MIRROR ZFS

Il primo esempio concreto riguarda la transizione da array MD/RAID1 a MIRROR ZFS su Linux. Un MIRROR ZFS ha diversi vantaggi rispetto ad un array MD/RAID1: checksum di dati e metadati, resilvering più veloce, compressione LZ4 trasparente, crittografia nativa, possibilità di spostare i metadati (e dataset particolari) su unità a stato solido, ecc. Ho personalmente eseguito in passato sul mio PC la procedura di seguito descritta, per abbandonare MD/RAID1 a favore del MIRROR ZFS.

NOTA: Si ricorda che in Linux i nomi dei device non sono necessariamente persistenti, pertanto verificare sempre su quale device si sta operando. Eventualmente, è sempre possibile specificare un device a blocchi utilizzando il percorso "/dev/disk/by-path/<device>". Di seguito, per semplicità e compattezza, utilizzeremo i nomi classici.

Premessa: l'array MD/RAID1 di partenza, denominato */dev/md127*, è composto dalle partizioni */dev/sdc1* e */dev/sdd1* di due HDD WD GOLD 8TB, ed è inizialmente montato in */mnt/STORAGE*.

NOTA: al tempo della conversione non avevo un disco "spare" di opportuna capacità, ma solo una copia dei dati più importanti; ho quindi rischiato, procedendo alla transizione eliminando prima uno dei dischi dall'array MD/RAID1. Ciò ha comportato la perdita della ridondanza (ma è andato tutto bene). In genere, tale situazione è da evitare, a meno che non si abbia un backup aggiornato dell'intero raid (che è sempre auspicabile). Nel prossimo esempio vedremo come espandere il pool ZFS senza rinunciare alla ridondanza.

Il pool ZFS è denominato "*storage*" e viene montato nella stessa posizione del vecchio RAID1 (/mnt/STORAGE). Le unità di storage utilizzate per il pool ZFS sono le seguenti:

- SATA3 (/dev/sdc) & SATA4 (/dev/sdd) : HDD WD GOLD 8TB → ZFS pool principale.
- SATA1 (/dev/sda) & SATA2 (/dev/sdb) : SSD KINGSTON DC500M 960GB → ZFS special device;

Gli HDD WD da 8TB sono quelli su cui è presente l'array MD/RAID1 di partenza, mentre gli SSD KINGSTON sono aggiunti appositamente per ZFS. Se si desidera ripercorrere la procedura si raccomanda la massima attenzione, in quanto alcuni comandi possono distruggere il pool o altre partizioni se si commettono errori:

1. Ripristinare uno stato pulito del sistema con l'applicazione "timeshift" (se necessario), quindi installare i moduli ZFS per il kernel corrente ("linux*-zfs", "zfs-kmod" o altro pacchetto in base alla distribuzione utilizzata): il pacchetto "zfs-utils" (o "zfs") dovrebbe essere installato come dipendenza. Riavviare e verificare che il comando "zpool status" funzioni senza errori (anche se non restituisce nulla).
2. Commentare /mnt/STORAGE in /etc/fstab (come visto, con ZFS non serve).
3. Dissociare /dev/sdc1 da MD/RAID1:

```
mdadm /dev/md127 --fail /dev/sdc1
mdadm /dev/md127 --remove /dev/sdc1
```

4. Eliminare i metadati MD/RAID1 e la tabella delle partizioni da /dev/sdc:

```
wipefs -a /dev/sdc1 /dev/sdc
```

5. Smontare /mnt/STORAGE (punto di mount iniziale di MD/RAID1) e rimontare manualmente l'array /dev/md127 (ora degradato) in una posizione diversa (ad esempio /root/md127).
6. Creare lo zpool "storage" completo di special device, specificando il WD GOLD rimosso da MD/RAID1 (/dev/sdc) e uno dei due KINGSTON (per maggiore sicurezza, utilizzare /dev/disk/by-path/<device>):

```
zpool create -m /mnt/STORAGE -o ashift=12 storage /dev/sdc special /dev/sda
```

7. Verificare che il filesystem ZFS sia stato creato e montato:

```
zpool status
zpool list -v
zfs list
mount | grep storage
```

8. Impostare alcune proprietà del pool per ottimizzare le prestazioni:

```
zfs set recordsize=1M storage
zfs set compression=lz4 storage
zfs set atime=off storage
zfs set xattr=sa storage
```

9. Verificare le proprietà appena impostate:

```
zfs get all storage | grep -w -E "xattr|atime|compression|recordsize"
```

10. Creare un dataset radice per ogni utente a cui è concesso memorizzare dati sul pool ZFS: avere dataset separati per ogni utente consente di applicare diverse politiche di snapshot, impostare lo spazio massimo che ciascun utente può utilizzare (quote), ottimizzare la dimensione dei record per dataset, ecc:

```
zfs create storage/<user1>
zfs create storage/<user2>
...
```

    Se necessario, abilitare la crittografia contestualmente alla creazione di tali dataset (fare riferimento alla sezione "26 Native encryption"), fornendo una password opportuna per ogni utente creato.

11. Se necessario imporre delle quote allo spazio ZFS riservato ai vari utenti:

```
zfs set quota=<size1> storage/<user1>
zfs set quota=<size2> storage/<user2>
...
```

12. Per ogni utente aggiunto al pool ZFS, creare i dataset figli, imitando la struttura delle cartelle principali (top) eventualmente presenti (per il corrispettivo utente) nell'array MD/RAID1; rispetto all'utilizzo di un unico dataset per utente, ciò ha il vantaggio di poter applicare proprietà diverse ai diversi dataset figli (come vedremo, sfrutteremo tale caratteristica per spostare alcuni dataset sullo special device, in modo che questi abbiano un accesso più veloce, essendo su SSD anziché HDD):

```
for i in AUDIO BACKUP DOCUMENTI EMAIL ...; do zfs create storage/<user1>/$i; done
...
```

    Si può anche procedere oltre nella gerarchia, per specializzare ulteriormente le proprietà dei dataset.

13. Verificare i dataset creati:

```
zfs list
```

14. Assegnare proprietario e gruppo ai vari dataset di ciascun utente creato:

```
chown -R <user1>:<user1> /mnt/STORAGE/<user1>
chown -R <user2>:<user2> /mnt/STORAGE/<user2>
...
```

15. Stabilire quali dataset devono essere completamente allocati sullo special device (fare attenzione alle dimensioni occupate); ad esempio, se i dataset DOCUMENTI ed EMAIL non contengono grossi file (come video, VM, ...), essi possono essere allocati sullo special device per un accesso più veloce:

```
zfs set special_small_blocks=1M storage/<user1>/DOCUMENTI
zfs set special_small_blocks=1M storage/<user1>/EMAIL
...
```

16. Spostare i dati da MD/RAID1 a MIRROR ZFS (utente → utente, cartella top → dataset):

```
mv /root/md127/<user1>/AUDIO/* /mnt/STORAGE/<user1>/AUDIO/
...
```

NOTA: Se si è creata una gerarchia di dataset, è bene iniziare lo spostamento da quelli più profondi, eliminando ogni volta la directory sorgente quando tale spostamento è terminato (per rimuovere una cartella, utilizzare sempre il comando "rmdir" e non "rm -rf", in tal modo se qualcosa non è stato spostato non si correrà il rischio di perdere file); notare che è in questa fase che i vari dataset vengono popolati secondo le configurazioni impostate (compressi, criptati, sul device speciale, ecc).

17. Disassemblare completamente l'array MD/RAID1:

```
mdadm --stop /dev/md127
```

18. Eliminare i metadati MD e la tabella delle partizioni del secondo disco in precedenza appartenuto all'MD/RAID1:

```
wipefs -a /dev/sdd1 /dev/sdd
```

19. Creare il mirror per lo "special device" (aggiungendo il secondo SSD KINGSTON):

```
zpool attach -o ashift=12 storage /dev/sda /dev/sdb
```

20. Creare il mirror per il pool principale (aggiungendo il secondo HDD WD GOLD):

```
zpool attach -o ashift=12 storage /dev/sdc /dev/sdd
```

21. Abilitare i servizi necessari affinché il pool ZFS venga riconosciuto e montato automaticamente al riavvio del sistema (fare riferimento alla documentazione della distribuzione utilizzata):

```
systemctl enable zfs-import-cache
systemctl enable zfs-import.target
systemctl enable zfs-mount
systemctl enable zfs.target
```

22. Riavviare il PC, verificare che il pool ZFS sia stato riconosciuto e montato (con "zpool status"); se tutto è ok, effettuare il backup del sistema con timeshift (per il salvataggio delle modifiche ad /etc).

29.2 Espansione del pool "storage" (MIRROR ZFS su SATA)

Nella sezione "2 Zpool, vdev e dataset" abbiamo visto che quando l'occupazione di un pool ZFS raggiunge l'80%, è necessario espandere il pool per evitare problemi futuri. Questa espansione può avvenire in due modi:

- aggiungendo un nuovo vdev al pool (quando possibile e conveniente);
- espandendo uno dei vdev già esistenti, di cui il pool è costituito.

In un generico vdev, l'espansione si esegue sostituendo le unità di storage una alla volta, e attendendo il completamento del resilvering per ciascun disco. Al termine dell'operazione, il pool risulterà automaticamente espanso. Va notato che, durante l'espansione di un vdev, se quest'ultimo è configurato come MIRROR o RAIDZ1, il vdev (e quindi il pool) rimane privo di ridondanza per tutta la durata del processo. Questo rende l'operazione molto rischiosa: se si verifica un errore (o un danno hardware), il pool potrebbe essere irreparabilmente danneggiato. Invece, configurazioni RAIDZ2 e RAIDZ3 offrono una maggiore garanzia grazie alla ridondanza residua (doppia nel caso di RAIDZ3), ma è importante ricordare che queste configurazioni impongono un carico maggiore sullo storage durante il resilvering.

Comunque, in un vdev configurato come MIRROR, è possibile espandere il pool mantenendo la ridondanza, usando una procedura leggermente diversa. Ricordando che un MIRROR può essere composto da un numero N qualsiasi di dischi (con $N \geq 2$), basta inizialmente aggiungere un nuovo disco al mirror (anziché sostituirlo), attendere il completamento del resilvering, quindi sostituire uno dei vecchi dischi, ripetere il resilvering e al termine eliminare l'ultimo vecchio disco rimasto. In questo modo, la ridondanza è preservata.

Per mantenere la ridondanza è dunque necessario disporre di una connessione SATA libera e spazio nel "case" per alloggiare temporaneamente il disco aggiuntivo (qualora si disponesse di due connessioni SATA libere e

spazio per due dischi aggiuntivi, è possibile espandere il pool ZFS con un solo resilvering, dimezzando i tempi). Se non si hanno connessioni SATA libere, è possibile utilizzare una scheda di espansione PCIe-SATA economica. Il problema potenzialmente maggiore è quindi rappresentato da un "case" inadeguato, con spazio insufficiente al montaggio di dischi aggiuntivi. In tal caso, è possibile "appoggiare" i dischi esternamente, ma questa soluzione comporta notevoli rischi: il resilvering può durare molte ore e un piccolo urto alle unità di storage potrebbe compromettere l'intera operazione. Una possibilità alternativa è quella di utilizzare due bridge USB3-SATA o, meglio ancora, un DAS (Direct Attached Storage), cioè un bridge USB3-SATA che supporti due o più dischi. L'unica condizione necessaria per garantire il successo dell'operazione è che i bridge usati siano di ottima qualità (ricordare che ZFS non tollera bene gli errori di I/O); inoltre, si raccomanda l'uso di alcool isopropilico prima di iniziare l'operazione (per garantire la massima affidabilità delle connessioni USB3).

Quest'ultimo è l'approccio che utilizzeremo nel nostro esempio pratico. Vedremo quindi come espandere il MIRROR costituito dai due HDD WD GOLD 8TB sostituendoli con due HDD WD Ultrastar 22TB, utilizzando temporaneamente un DAS QNAP TR-002 come supporto (notare che il device speciale non viene toccato). Il motivo di tale scelta è che avevo già il QNAP TR-002 per l'uso come unità di backup (con due dischi da 3TB). Mi è dunque bastato estrarre temporaneamente le due unità da 3TB per avere il DAS a disposizione.

I passi per l'espansione del mirror sono i seguenti:

1. Effettuare eventuali aggiornamenti del firmware nel bridge USB3-SATA utilizzato (in questo esempio il QNAP TR-002); è bene che il fw sia la versione più aggiornata.
2. Effettuare lo scrub dello zpool (è sempre bene partire da una situazione "pulita").
3. Disattivare i servizi ZFS (da root); nel caso di servizi "systemd" disabilitare e fermare:
   - zfs.target (Target di avvio ZFS)
   - zfs-import-cache.service (Importa pool ZFS mediante /etc/zfs/zpool.cache)
   - zfs-import-scan.service (Importa pool ZFS mediante scansione)
   - zfs-import.target (Target di importazione pool ZFS)
   - zfs-mount.service (Montaggio automatico di filesystem ZFS)
   - zfs-zed.service (Demone ZFS degli eventi)
   - zfs-trim-weekly@storage.timer (Trim settimanale del pool ZFS "storage")
4. Mascherare il servizio "zfs-import-scan.service" (su alcune distro Linux, come Gentoo, tale servizio potrebbe comunque essere avviato automaticamente, se non mascherato, importando il pool).
5. Eseguire:

```
zpool export storage        (ove 'storage' è il nome del pool ZFS)
dracut --force              (su Gentoo e altri OS che usano dracut per initramfs)
reboot
```

6. Entrare come root da console VT (quindi NON come utente in ambiente grafico).
7. Verificare che il pool non sia stato reimportato.
8. Effettuare lo shutdown.
9. Smontare i due dischi magnetici WD GOLD 8TB e montare al loro posto i nuovi dischi WD Ultrastar 22TB (prima rimuovere la polvere presente all'interno del case del PC e pulire bene i contatti dei connettori SATA e POWER con alcool isopropilico). Non fornire ancora alimentazione al PC.
10. Montare i due dischi WD GOLD 8TB nel DAS QNAP TR-002:
    - rimuovere i dischi presenti nel QNAP TR-002 (o altro DAS utilizzato) e riporli in luogo riparato;
    - assicurarsi che i dip switch siano posizionati tutti su OFF (modalità "dischi singoli");
    - utilizzare alcool isopropilico per pulire i connettori SATA e POWER degli HDD;
    - inserire i due dischi WD GOLD 8TB negli slot del QNAP (estraendoli e reinserendoli un paio di volte, per eliminare eventuali patine d'ossido dai contatti);

- pulire con alcool isopropilico i connettori USB3 del QNAP, del cavo dati e del PC;
- per evitare problemi durante il resilvering, connettere il QNAP alla stessa sorgente di alimentazione del PC, attraverso un UPS (o centralina elettrica): in caso di caduta di alimentazione, qualora la tensione di rete non torni velocemente, sarà possibile effettuare lo shutdown regolare e il resilvering riprenderà al momento che l'alimentazione sarà ripristinata, il PC riavviato e il pool reimportato.

NOTA: ZFS traccia i dischi appartenenti ad un pool mediante un identificativo della partizione GPT, e non tramite la posizione sul controller SATA (o USB3-SATA), quindi lo spostamento dei dischi da SATA a USB3-SATA non comporta problemi, a patto di effettuare prima l'export (e poi l'import) del pool e pulire ben bene tutti i contatti SATA ed USB (si ricorda che ZFS non tollera bene gli errori di I/O).

11. Ora fornire alimentazione al PC e importare il POOL:
    - dopo il boot di Linux, non entrare come utente ma come root su VT;
    - accendere il QNAP senza cavo USB connesso; quando i dischi sono andati in stand-by connettere il cavo USB al PC: i dischi dovrebbero avviarsi; attendere l'accensione della spia blue sul QNAP (indica che l'unità è pronta) e verificare con il comando "dmesg' che i dischi siano visti da Linux;
    - verificare che il pool sia riconosciuto (senza importarlo) digitando:

      ```
      zpool import -d /dev/disk/by-path
      ```

      Se tutto è ok, si può proseguire, altrimenti risolvere prima eventuali problemi.
12. Se desiderato, in questa fase, è possibile modificare la struttura dei settori logici (da 512e a 4K) sui nuovi dischi magnetici (i due WD Ultrastar 22TB montati internamente al PC)[30]:

    ```
    hdparm --set-sector-size 4096 --please-destroy-my-drive /dev/sdc
    hdparm --set-sector-size 4096 --please-destroy-my-drive /dev/sdd
    reboot  (IMMEDIATO, NON FARE ALTRO PRIMA DEL REBOOT)
    ```

    NOTA1: verificare tramite comando "smartctl -a" che i settori 4K siano disponibili nei nuovi dischi; tutte le unità di storage magnetiche prodotte negli ultimi 10 anni hanno settori fisici 4K, ma di solito solo le unità di classe "enterprise/datacenter" permettono di modificare la dimensione dei settori logici. Il vantaggio dei settori 4K è che sfruttano al massimo il throughput del bus SATA.

    NOTA2: la modifica della dimensione dei settori logici distrugge le partizioni, con conseguente perdita di eventuali dati presenti; fare attenzione a specificare i device corretti.

    NOTA3: non eseguire tale conversione sui vecchi dischi (montati nel DAS): molti bridge USB3-SATA (incluso il QNAP TR-002) non supportano la doppia modalità 512e/4K, ma forzano l'una o l'altra (molto più spesso la 512e, come appunto avviene nel QNAP), né supportano i comandi per tale conversione, per cui eseguire questa procedura su dischi connessi via USB può causare malfunzionamenti o portare al blocco irrimediabile delle unità di storage. Per verificare se il QNAP (o altro bridge USB3-SATA utilizzato) supporta i settori logici 4K basta utilizzare il comando "smartctl -a" (sui relativi device).
13. Se si è effettuata la conversione a 4K nei nuovi dischi (montati internamente al PC), verificare con "gdisk" e "smartctl -a" che settori logici/fisici siano entrambi 4K.
14. Mediante gdisk, creare la tabella GPT delle partizioni (senza creare alcuna partizione) su tali dischi.
15. Effettuare un test SMART veloce su di essi:

    ```
    smartctl -t short /dev/sdc
    smartctl -t short /dev/sdd
    ```
16. Importare il pool, verificando prima che il mirror veda i dischi USB esterni:

    ```
    zpool import -d /dev/disk/by-path
    zpool import -d /dev/disk/by-path storage
    ```

    Se è tutto ok, si può uscire da root ed entrare come utente in KDE/PLASMA (o altro ambiente utilizzato).

[30] Vedere https://wiki.archlinux.org/title/Advanced_Format

17. Provare a copiare un grosso file dal pool (ad esempio un disco di macchina virtuale) per assicurarsi che non ci siano errori di I/O sul bus USB (in caso di errori di I/O effettuare immediatamente lo shutdown tramite "Magic SysRq Key" + R.E.I.S.U.B., pulire i contatti e riprovare). Viceversa, se tutto è ok, digitare:

```
zpool set autoexpand=on storage
zpool attach -o ashift=12 storage /dev/sdf /dev/sdc  (con /dev/sdf disco1 del DAS USB)
zpool attach -o ashift=12 storage /dev/sdg /dev/sdd  (con /dev/sdg disco2 del DAS USB)
```

NOTA1: il primo comando è necessario per attivare l'autoespansione del pool.

NOTA2: il parametro "ashift" va sempre specificato, e deve concordare con quello impostato in fase di creazione del pool (utilizzare "zpool get ashift storage" per verificarlo).

NOTA3: /dev/sdf e /dev/sdg sono solo di esempio; per maggiore sicurezza si consiglia di utilizzare /dev/disk/by-path/<device> (in tal modo è evidente quali device sono sul bus USB).

NOTA4: ZFS crea di default 2 partizioni su ogni disco; la seconda (n.9) è riservata, e creata per tenere conto di lievi differenze tra dischi con stessa capacità dichiarata ma di produttori diversi; la dimensione della partizione n.9 dipende dalla capacità totale del disco (di solito è 8MB ma può arrivare a 64MB per dischi molto grandi), ma è comunque sempre trascurabile rispetto alle dimensioni totali del disco. È possibile espandere il pool per usare anche tale spazio riservato[31], ma è sconsigliato farlo, poiché in caso di failure di uno dei dischi del pool, ciò costringerà ad acquistare esattamente lo stesso modello che si è guastato (se è ancora in commercio) oppure uno con capacità sicuramente maggiore (ad es. il taglio successivo); visto l'esiguo spazio sprecato, non vale la pena rischiare possibili complicazioni.

18. Attendere il resilvering (ci possono volere molte ore, in funzione di quanti dati sono presenti). Al termine si sarà ottenuto un mirror a 4 dischi.
19. Effettuare lo scrub del pool (questo dovrebbe impiegare circa la metà del tempo che è stato necessario per il resilvering).
20. Al termine dello scrub possiamo eliminare i vecchi dischi dal pool (quelli contenuti nel DAS USB):

```
zpool detach storage /dev/sdf
zpool detach storage /dev/sdg
```

NOTA: attenzione a non effettuare il detach dei nuovi dischi (altrimenti occorrerà ricominciare tutto da capo), né quelli utilizzati per lo special device (altrimenti metadati e altri file presenti sui due SSD verranno spostati sulle unità magnetiche).

21. Verificare con "zpool list -v" che il pool è stato automaticamente espanso, quindi esportarlo.
22. Digitare: poweroff
23. Disconnettere il cavo USB del QNAP TR-002 (e rimontare i dischi originali al suo interno).
24. Entrare come root in console VT e digitare:

```
zpool status                                    (dovrebbe essere vuoto)
zpool import -d /dev/disk/by-partuuid/          (deve mostrare la struttura del pool)
zpool import -d /dev/disk/by-partuuid/ storage  (riassembla il pool usando i partiiud)

systemctl unmask zfs-import-scan.service
systemctl enable zfs-import-cache.service       (usare 'import-cache.service' oppure
                                                'import-scan.service' (scelta libera))
systemctl enable zfs.target
systemctl enable zfs-mount.service
systemctl enable zfs-import.target
systemctl enable zfs-trim-weekly@storage.timer
systemctl enable zfs-zed.service

systemctl start zfs.target
dracut --force
reboot
```

[31] Vedere https://serverfault.com/questions/946055/increase-the-zfs-partition-to-use-the-entire-disk

29.3 Espansione del pool "zfsbackup" (MIRROR ZFS su DAS USB)

Come ho reimpiegato i due dischi WD GOLD 8TB rimossi dal PC? Ovviamente espandendo il pool di backup presente sul QNAP TR-002, originariamente basato su due WD RED 3TB. Tali unità avevano passato i dieci anni di onorato servizio (erano i vecchi modelli WD RED CMR, paragonabili ai recenti WD RED PRO) e pur continuando a funzionare perfettamente, ho ritenuto utile sostituirle con i WD GOLD 8TB, che oltre a garantire maggiore capacità offrono prestazioni superiori grazie alla loro tecnologia più recente.

In tal caso non è stato possibile preservare la ridondanza del pool "zfsbackup" (ma poco male, il pool "storage" era ormai al sicuro, se l'espansione di "zfsbackup" fosse fallita avrei potuto ricostruire tale backup).

Ecco i passi da me seguiti per tale espansione (notare che se il pool è criptato, non occorre decrittarlo):

1. Sostituire uno dei WD RED 3TB nel DAS con un WD GOLD 8TB (notare che entrambi sono dischi CMR; gli SMR non vanno bene per il RAID, indipendentemente che sia di tipo software o hardware).
2. Accendere il DAS, attendere lo stand-by dei dischi e connetterlo mediante cavo USB al PC, quindi attendere il riconoscimento delle unità di storage.
3. Creare una nuova tabella GPT sul disco WD GOLD 8TB (attenzione a specificare il device corretto):

   ```
   gdisk /dev/sdf      (o -> w -> q)
   ```

4. Importare il pool "zfsbackup":

   ```
   zpool import -d /dev/disk/by-path/
   zpool import -d /dev/disk/by-path/ zfsbackup    (se fallisce, aggiungere l'opzione '-f')
   zpool status                                    (uno dei dischi risulterà in stato FAULTED)
   ```

   Fare caso al codice numerico del disco FAULTED riportato da "zpool status"; indicheremo tale codice con "<id_faulted_disk>" (ci servirà a breve);

5. Ora digitare:

   ```
   zpool set autoexpand=on zfsbackup
   zpool get autoexpand zfsbackup                  (verifica autoexpand)
   zpool get ashift zfsbackup                      (verifica ashift=12 o superiore)
   ```

6. Comunicare a ZFS il disco da sostituire al posto di quello FAULTED:

   ```
   zpool replace -o ashift=12 -f zfsbackup <id_faulted_disk> /dev/sdf
   ```

   ove <id_faulted_disk> è il codice numerico riportato da "zpool status";

7. Attendere il resilvering.
8. Verificare lo stato del pool ed esportarlo.

   ```
   zpool status
   zpool export zfsbackup
   ```

9. Effettuare lo shutdown.
10. Ripetere quanto fatto per il secondo WD GOLD 8TB (passi da 1 a 7).
11. Al termine, verificare con "zpool list -v" che il pool è stato espanso.
12. Esportare il pool e reimportarlo tramite il metodo usualmente adottato per l'accesso a "zfsbackup" (nel mio caso utilizzo uno script bash per l'importazione e la decrittazione mediante password).
13. Effettuare uno scrub finale.

30 Riferimenti e altri link utili

- *https://wiki.gentoo.org/wiki/ZFS*
- *https://wiki.archlinux.org/title/ZFS*
- *https://blog.victormendonca.com/2020/11/03/zfs-for-dummies/*
- *https://www.ixsystems.com/documentation/freenas/11.3-U5/zfsprimer.html*
- *https://www.reddit.com/r/zfs/comments/lyanut/zfs_recommendations_for_new_users/*
- *https://jrs-s.net/2018/08/17/zfs-tuning-cheat-sheet/*
- *https://jrs-s.net/2018/04/11/primer-how-data-is-stored-on-disk-with-zfs/*
- *https://openzfs.github.io/openzfs-docs/Project%20and%20Community/FAQ.html*
- *https://openzfs.github.io/openzfs-docs/Performance%20and%20Tuning/Workload%20Tuning.html*
- *https://www.truenas.com/community/threads/zil-and-l2arc-on-same-disk.23333/*
- *https://serverfault.com/questions/1000767/ext4-vs-xfs-vs-Btrfs-vs-zfs-for-nas*
- *https://askubuntu.com/questions/87035/how-to-check-hard-disk-performance*
- *https://www.truenas.com/wp-content/uploads/2023/11/ZFS_Storage_Pool_Layout_White_Paper_November_2023.pdf*

30.1 Pagine man principali

- zpoolconcepts(7)
- zpoolprops(7)
- zpool-features(7)
- zpool-create(8)
- zpool-add(8)
- zpool-attach(8)
- zpool-detach(8)
- zpool-online(8)
- zpool-remove(8)
- zpool-destroy(8)
- zpool-import(8)
- zpool-status(8)
- zfsprops(7)
- zfs-list(8)