ZFS in Linux

v1.3

Concetti di base

ZFS (Zettabyte File System)[1] è un filesystem a 128bit open source, 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, 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. 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 installata). Molte distribuzioni Linux forniscono nei loro repository un modulo “zfs” precompilato per ogni versione di kernel supportata.

(Z)pool, vdev e dataset

Prima di vedere i comandi principali per operare con il filesystem ZFS occorre comprendere i concetti di “pool” (o zpool), “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” (con due o più dischi per vdev) mentre tra i vdev del pool è attivo il “data striping”. Lo spazio complessivo, fornito dai vdev, è reso disponibile a tutti gli oggetti del pool (“dataset”). Si possono avere quattro tipologie di “dataset”:

1 https://it.wikipedia.org/wiki/ZFS_(file_system)

- “*ZFS Filesystem*” : classica organizzazione dei dati in file e cartelle, con accesso mediante semantica POSIX;

- “*ZFS Volume*” : volume logico 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 punto seguente);

- “*ZFS Clone*” : copia di uno snapshot (filesystem o volume) che è possibile modificare;

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, ecc.

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

- Al 90% della capacità di storage, ZFS passa da un'ottimizzazione "basata sulle prestazioni" ad un'ottimizzazione "basata sullo spazio", con conseguente rallentamento. Per ottenere le massime prestazioni in scrittura e per evitare problemi con la eventuale 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 dei dischi 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 è 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. Se il numero di dischi persi dal vdev (incluso quello in ricostruzione) supera il numero consentito, i dati nel pool vanno persi. Per questo motivo, lo schema RAIDZ1 è sconsigliato per unità di dimensioni superiori a 1 TB.

- Il MIRROR consuma più spazio su disco 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 più avanti).

- All’interno di un vdev si dovrebbero utilizzare dischi con caratteristiche molto simili (preferibilmente tutti dello stesso modello, ma appartenenti a lotti di produzione diversi).

NOTA: qualora ZFS venga utilizzato con un controller RAID hardware, è fortemente consigliato impostare quest’ultimo in modalità JBOD, affinché ZFS abbia il pieno controllo dei singoli dischi.

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.

VDEV a singolo disco

Un pool di più vdev, formati ciascuno da un singolo disco, esprime una struttura che ricorda vagamente il RAID0, sebbene non sia un vero RAID0. Le scritture sono infatti distribuite tra i vdev del pool in base al relativo spazio libero e non mediante “stripe” uniformi come avviene con il

classico RAID0 (ZFS non prevede il RAID0, in quanto non conforme con lo scopo primario del filesystem che, come detto, è quello di preservare l'integrità dei dati):

La capacità di storage di questa struttura è pari alla somma delle capacità dei singoli dischi, 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 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 dello zpool 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[2]. 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 512B (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 occorre usare ashift=12 o ashift=13[3].

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 specificare dei file come device. Ad esempio, 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 La dimensione dei settori fisici di un HDD si può leggere in Linux con il comando "fdisk -l" o "hdparm -I". Per gli SSD non è così semplice: il firmware spesso mente riportando 512B ma tutti gli SSD hanno settori da almeno 4KB.

3 https://forum.proxmox.com/threads/samsung-ssds-a-right-ashift-size-for-zfs-pool.71627/

VDEV ridondanti (MIRROR)

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

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 viste in precedenza, che andranno comunque 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 dal singolo vdev)
zpool attach tank /dev/sda /dev/sdb	(aggiunge il secondo device creando il mirroring)

Ciò è utile ad esempio nella transizione da un RAID1 classico a MIRROR ZFS: basta marcare come “faulty” uno dei due dischi del RAID1, eliminarlo dall’array e riassegnarlo ad uno pool ZFS. Il RAID1 resterà in piedi (anche se degradato) e sarà possibile effettuare la copia 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 vanno 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 con “zpool create”.

Una configurazione più interessante è quella relativo ad un pool costituito da più vdev, ove ciascuno è un mirror di due 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 più avanti le considerazioni sul pool 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 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 gli altri dispositivi:

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 assomiglia ad un RAID10 ma non lo è: 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 maggiori viene sprecato e lo spazio di archiviazione totale è di 2 TB; viceversa, mettendo le stesse unità in uno zpool di due MIRROR, si ottiene un MIRROR da 1 TB più un MIRROR da 2 TB e lo spazio di archiviazione totale è di 3 TB. Come detto, le scritture verranno 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: quando viene scritto un solo record (vedere "recordsize" più avanti) 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, per cui è sempre consigliabile utilizzare vdev con stessa struttura e dischi con caratteristiche molto simili (possibilmente stessa marca e modello, ma lotti diversi per ridurre la possibilità che un eventuale difetto di fabbrica sia presente in più di una unità).

VDEV con parità singola (RAIDZ1)

La configurazione di base è quella mostrata in figura:

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 raidz[4] (così come i dati veri e propri).

Ovviamente nulla vieta di creare un pool costituito da più vdev RAIDZ1. Ad esempio, dodici dischi possono essere suddivisi in due vdev RAIDZ1 di sei dischi ciascuno; in tal modo le scritture verranno bilanciate sui vdev del pool, aumentando le prestazioni del filesystem, analogamente a quanto avviene per il pool costituito da due o più MIRROR.

La creazione di un pool con vdev RAIDZ1 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:

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.

VDEV con parità doppia e tripla (RAIDZ2 e RAIDZ3)

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

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”.

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: 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 danneggerà l'intero pool.

4 Vedere la pagina man zpoolconcepts(7)

Considerazioni sullo stato degradato

Potrebbe essere allettante optare per la configurazione che possiede l'efficienza di storage più alta (RAIDZ1), ma ciò è abbastanza rischioso. Se un disco si guasta, le prestazioni del pool saranno drasticamente ridotte durante la sostituzione e il resilvering. Inoltre durante 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.

RAIDZ2 / RAIDZ3 costituiscono un rimedio parziale a questo scenario catastrofico in quanto utilizzano doppia / tripla parità, tollerando rispettivamente due / tre dischi guasti. Il problema però è solo mitigato, perché le prestazioni di un pool RAIDZ2 / RAIDZ3 degradato e il tempo di resilvering sono peggiori di un RAIDZ1, perché i calcoli di parità sono considerevolmente più complicati (il peggioramento è tanto più evidente quanto maggiore è il numero di dischi nel vdev).

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 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 / RAIDZ3; inoltre esso causa un impatto minore sulle prestazioni del pool. Notare che in un pool di MIRROR la resilienza a più guasti è comunque forte; ad esempio, con 8 dischi di stessa marca e modello (4 mirror di 2 dischi ciascuno) si ha:

- 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% → probabilità di sopravvivenza = 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)) * 100*, ove 'f' è il numero di dischi già guasti ed '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).

La 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, e molto più veloce.

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

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 conviene utilizzare vdev RAIDZ1 da 3 dischi, RAIDZ2 da 6 e RAIDZ3 da 9, in modo da riservare sempre ⅓ 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.

In conclusione conviene usare i vdev MIRROR, a meno di non essere ben consci di cosa si sta facendo e delle implicazioni future.

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 (viene eseguita la migrazione dei dati al nuovo dispositivo e al termine il vecchio è rimosso dalla configurazione).

Infine è possibile rimuovere vdev a singolo disco o interi vdev ridondati (MIRROR) con il comando "zpool remove" (da OpenZFS v0.8 in poi):

zpool remove <pool> <vdev>

ZFS tenterà di spostare gli eventuali dati presenti nel vdev che si desidera rimuovere sugli altri vdev del pool.

Attualmente (OpenZFS ≤ v2.2.6) l'operazione "remove" non è permessa su pool che contengono vdev con parità (RAIDZ), qualsiasi siano gli eventuali altri vdev che costituiscono il pool. Se per errore si aggiunge un singolo disco al pool, poi non è più possibile rimuoverlo.

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

Disabilitare l'access time (atime)

Non è necessario tracciare ora e data di accesso ai file a meno di scenari altamente specifici. La disabilitazione di atime riduce le scritture sul filesystem migliorando le prestazioni di I/O (e qualora il pool sia basato su SSD aumenta la durata delle memorie flash da essi utilizzati):

zfs set atime=off <pool>

Per verificare che atime sia stato disabilitato digitare:

zfs get all <pool> | grep time
mount | grep <pool>

Modificare lo storage degli attributi estesi (xattr)

A differenza di Solaris, OpenZFS su Linux di default memorizza gli attributi estesi (in gergo "xattr" o "NFSv4 ACL") in sottodirectory nascoste, causando molteplici accessi al filesystem. Conviene invece memorizzare tali attributi negli inode del filesystem, per generare minori richieste di I/O:

zfs set xattr=sa <pool>

Per verificare che xattr sia stato imposto correttamente digitare:

zfs get all <pool> | grep xattr

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[5], per cui non ci sono controindicazioni ad abilitare tale compressione di default sul pool:

zfs set compression=lz4 <pool>

5 https://klarasystems.com/articles/openzfs1-understanding-transparent-compression/

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 su cosa si sta facendo. 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 nella pagina man zfsprops(7).

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. Tale dimensione deve essere necessariamente una potenza di 2. Ad esempio, per impostare una dimensione massima di 1 megabyte 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 (i motivi di ciò 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. Ciò 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 conveniente (sempre di dimensione pari a una potenza di 2) evitando inutili sprechi di spazio all'interno del pool. Il seguente comando permette di vedere alcune statistiche e altre informazioni sui blocchi allocati dall'intero pool:

zdb -Lbbbs <pool>

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

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 l'encryption nativo 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 hash 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 abbastanza inutile su vdev ridondanti (mirror) o vdev con parità.

Notare che una corruzione silenziosa è impossibile in ZFS, poiché tutto viene sottoposto ad hashing anche con "copies=1" (default) 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 ricostruiti per mancanza di ridondanza.

Importazione/esportazione e mount del pool al riavvio

ZFS, durante la creazione del pool, effettua anche il montaggio del filesystem nel punto di mount specificato (o quello di default). Ma la rilevazione del pool, ed il montaggio del filesystem, non sopravvivono al riavvio del sistema. Per tale scopo occorre abilitare alcuni servizi. Con systemd[6]:

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

Con OpenRC[7]:

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

Esistono anche altri servizi relativi a ZFS, ad esempio la condivisione automatica su NFS o SMB (zfs-share) e la notifica via email di cambiamenti di stato (o altri eventi) associati ai filesystem ZFS (zfs-zed). Per questi servizi si rimanda alla relativa documentazione.

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:

6 https://wiki.archlinux.org/title/ZFS
7 https://wiki.gentoo.org/wiki/ZFS

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, quest’ultimo non verrà mostrato. In sostanza non si può fare affidamento sul file di cache. 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

(per zpool composti da vdev basati su file, basta specificare con l’opzione “-d” la directory in cui i vdev risiedono).

Il vantaggio dei due 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 tramite la loro posizione sul bus SATA o SCSI (*by-path*), piuttosto che tramite il classico nome di dispositivo (“sda”, “sdb”, ecc). Da parecchi anni, 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). È quindi molto utile la possibilità di identificare inequivocabilmente i device attraverso altri meccanismi. L’uso di nomi di dispositivi è comodo in fase di creazione del pool, ma poi conviene esportare e reimportare il pool utilizzando l’id o il path dei device: 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 molto 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, posizione sul bus e S/N. Prima di sostituire un disco, verificare sempre prima il SERIALE (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 danni involontari al pool.

NOTA: se si hanno più distribuzioni Linux da cui scegliere al boot, è possibile condividere il pool usando lo stesso file “*/etc/hostid*” per tutte le distro (altrimenti si avranno errori nell’importazione).

Rinominazione del pool

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

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 (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".

Creazione di volumi

Come accennato all'inizio, è possibile riservare spazio nel pool per creare device a blocchi ("volumi" in gergo ZFS). Un volume appare come device nella directory "/dev/zvol/". Ad esempio il seguente comando crea il volume "vol" di 2GB nel pool "tank":

zfs create -V 2GB tank/vol

Esso apparirà come device "*/dev/zvol/tank/vol*". 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; inoltre l'accesso a volumi "sparsi" è sicuramente meno efficiente di quello a volumi preallocati. L'uso più comune di volumi ZFS è come device di swap[8], ma esiste un bug relativo allo swap su ZFS, che causa una condizione di deadlock dell'OS (alla data corrente non ancora risolto[9]). Occorre inoltre 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[10].

Boot e root filesystem su ZFS, brevi considerazioni

FreeBSD ed alcune distribuzioni Linux supportano nativamente l'installazione su ZFS. Notare che 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, mentre usare un file (su filesystem ZFS) come swap non è possibile[11].

Espansione di vdev ridondanti e con parità

È possibile espandere facilmente la capacità di archiviazione di un vdev con parità sostituendo i dischi che lo compongono (come abbiamo visto non è possibile aggiungere dischi o mutare il tipo di parità dopo che il vdev è stato creato). Per prima cosa attivare l'auto espansione dei vdev del pool:

zpool set autoexpand=on <pool>

8 https://askubuntu.com/questions/228149/zfs-partition-as-swap
9 https://github.com/openzfs/zfs/issues/7734
10 https://openzfs.github.io/openzfs-docs/index.html
11 https://askubuntu.com/questions/1198903/can-not-use-swap-file-on-zfs-files-with-holes/1198916

Quindi sostituire un disco alla volta con le nuove unità di maggiore capacità, attendendo ogni volta il resilvering. Come discusso in precedenza, tale fase può richiedere molto tempo e comportare notevoli rischi, specie per vdev RAIDZ1. Comunque, al termine dell'operazione, ZFS espanderà automaticamente la capacità complessiva del vdev (e quindi del pool).

La cosa simpatica è che tale procedura funziona anche con i vdev ridondanti, per cui è molto semplice espandere un MIRROR con ZFS, ma anche più sicuro! Infatti, anche nel caso di un vdev MIRROR a due sole unità, è possibile mantenere la ridondanza dei dati: basta aggiungere (anziché sostituire) un nuovo disco al MIRROR, quindi procedere con la sostituzione di uno dei due dischi vecchi, e al termine dell'ultimo resilvering effettuare la rimozione del vecchio disco rimasto.

NOTA: prima di effettuare la sostituzione è bene stressare i nuovi dischi 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):

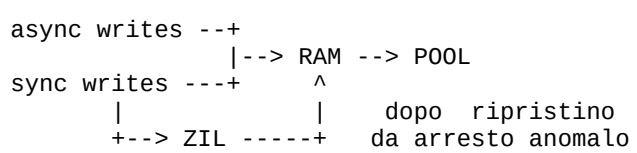
dd if=/dev/zero of=device bs=1MiB
smartctl -t long device

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, ed è anche quello di 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 fornisce anche la modalità di scrittura "sincrona" in cui, alla notificazione del completamento, il filesystem utilizzato deve garantire che una lettura successiva restituirà i dati corretti, indipendentemente da crash del sistema, riavvii, perdita di alimentazione, ecc. 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 sul disco. 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.

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 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 (dopo il reboot) i dati nello ZIL vengono 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 o NVMe di classe enterprise, utilizzando tale dispositivo come “registro di intenti separato” (SLOG):

```
async writes --+
               |--> RAM --> POOL
sync writes ---+     ^
       |             |    dopo  ripristino
       +--> SLOG ----+   da arresto anomalo
```

SSD o NVMe, 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:

```
              +--> RAM --> POOL
all writes -->|
              +--> ZIL or SLOG
```

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:

```
all writes --> RAM --> POOL
```

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 / NVMe 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 ha bisogno di avere grandi 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 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)

Con il comando "*zpool status*" è possibile verificare che l'SSD o l'NVMe sia stato messo in uso: il dispositivo SLOG viene riportato con il termine "log" nella lista dei componenti del pool. Ovviamente si può essere ancora preoccupati che i dati nello SLOG vadano persi se l'SSD o l'NVMe si guasta. In tal caso si possono utilizzare due SSD o NVMe in configurazione MIRROR:

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

È ovviamente anche possibile utilizzare più SSD o NVMe in configurazione RAIDZ ma ciò non ha molto senso poiché diminuisce le prestazioni dello SLOG senza apportare benefici.

NOTA: come visto in precedenza, ashift=12 o ashift=13 è raccomandato per gli SSD. Il firmware di molti SSD 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). Inoltre, come accennato sopra, si dovrebbero utilizzare SSD o NVMe di classe enterprise come SLOG ZFS, poiché quelli consumer non sopportano un numero abbastanza elevato di scritture esaurendosi rapidamente, hanno una maggiore latenza, e cosa ancora più importante non integrano un sistema di protezione dalla perdita di alimentazione (batteria tampone o condensatore 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 potenziali fault relativi all'alimentatore o all'UPS stesso, né alla rapida usura del disco a stato solido, la cui soluzione richiede necessariamente dispositivi di classe enterprise, che sono purtroppo molto più costosi.

Per verificare che 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!

ARC, L2ARC (cache)

Con ARC (Adaptive Replacement Cache) si intende una cache adattativa di lettura in RAM, per il filesystem e i dati di volume. ZFS utilizza sempre questa tipologia di cache. L2ARC è invece un livello di ARC che risiede su un sistema di archiviazione veloce, come un SSD o un disco NVMe. Una cache di secondo livello può sembrare allettante ma non sono tutte rose e fiori: innanzitutto,

anche se immagazzinata su SSD o NVMe, 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:

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

La cache L2ARC non può essere creata come mirror (non ha senso) ma può beneficiare di dispositivi multipli indipendenti (quindi dischi e non partizioni):

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

Un uso corretto di L2ARC richiede 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 un dispositivo cache raggiunga la piena funzionalità varia a seconda delle sue dimensioni. È possibile utilizzare il comando "zpool iostat" per monitorare la capacità utilizzata e le operazioni di lettura:

zpool iostat -v pool 5

Se si verifica un errore di lettura su un dispositivo L2ARC, la stessa operazione 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.

ZFS Metadata: special device

In ogni pool ZFS esiste una "classe speciale" di elementi formata da metadati, blocchi indiretti[12] ed eventuali tabelle di deduplicazione. Questa classe, può essere allocata su uno "special device" costituito da uno o più SSD / NVMe: 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.

12 https://en.wikipedia.org/wiki/Inode_pointer_structure

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 negli 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; ad esempio, per aggiungere al pool "tank" uno special device composto da 2 vdev MIRROR, realizzati con quattro NVMe, basta digitare:

zpool add tank special mirror /dev/nvme0n1 /dev/nvme1n1 mirror /dev/nvme2n1 /dev/nvme3n1

Si può anche procedere per passi successivi mediante "*zpool add*" e "*zpool attach*", come visto per i normali vdev (basta specificare il device già esistente al quale si vuole attaccare il nuovo device per rendere entrambi parte di un mirror). L'importante è utilizzare lo stesso tipo di unità di storage (per bilanciare bene l'I/O) e mantenere la coerenza del parametro "ashift".

Quanto spazio è necessario per lo special device? La regola empirica è quella di riservare almeno lo 0.3% della dimensione del pool normale. Ad esempio, per uno storage effettivo di 8TB, occorrono circa 25GB per la classe special: il resto è utilizzabile per piccoli blocchi di dati[13] o dataset particolari (vedere la prossima sezione). Di default, ZFS smette di allocare alcuni tipi di oggetti nello special device quando la sua occupazione arriva al 75% (il parametro è regolabile[14]): in tale eventualità, gli oggetti tornano ad essere memorizzati nel pool principale. Con uno storage di 8TB e special device di 960GB, restano 695GB che possono essere utilizzati per altro! (960*.75 – 25).

NOTA: ZFS non sposta mai deliberatamente dati o metadati già immagazzinati. Per spostare i metadati e altri oggetti 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.

Dati su special device

Per ciascun dataset è possibile determinare se ZFS debba allocare i blocchi di dati con dimensione minore o uguale a quella specificata sul pool principale o sullo special device:

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

Con il comando sopra si forza ZFS a memorizzare nella classe "special" (anziché nel pool principale) tutti i blocchi di dimensioni minori o uguali a 64KB e appartenenti allo specificato dataset. Se la dimensione specificata è quale alla dimensione massima ("recordsize") del dataset, allora tutti i blocchi di dati del dataset vengono allocati nello special device. In tal modo è possibile spostare interi dataset su SSD / NVMe 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 (normalmente nessun blocco di dati viene memorizzato nella classe special).

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

13 https://forum.level1techs.com/t/zfs-metadata-special-device-z/159954/65
14 https://forum.proxmox.com/threads/zfs-special-device-75.125956/

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. Per fortuna, esiste la possibilità di forzare esplicitamente un controllo di integrità sull'intero pool:

zpool scrub <pool>

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 (su unità SATA):

- pool su SSD 1TB → 15..20 minuti
- pool su mirror HDD 2 x 8TB + special device su mirror SSD 2 x 1TB → 4..5 ore
- pool su mirror HDD 2 x 3TB → 20..22 ore

Come si può osservare, lo scrub è particolarmente lento sui soli dischi magnetici, viene accelerato dalla presenza dello "special device" su SSD (grazie al fatto che almeno i metadati vengono spostati sulle unità a stato solido) ed è abbastanza veloce per pool interamente immagazzinati su unità FLASH. Ovviamente, a parità di hardware, maggiore è la dimensione del pool e la quantità di dati presenti, 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 (o schedularlo tramite timer systemd o cron) in orari o giorni in cui il sistema è sottoposto a minor carico. 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 + import), basta utilizzare il primo comando visto sopra. Per ulteriori informazioni si rimanda alla pagina man zpool-scrub(8).

ZFS Trim

È ben noto che i filesystem su SSD ed NVMe 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 ciò richiede del tempo; l'uso di TRIM permette di effettuare tali operazioni in 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[15]) in modo da incrementare la longevità delle unità di storage.

15 https://en.wikipedia.org/wiki/Wear_leveling

Generalmente, in Linux il trimming dei filesystem si effettua con "*fstrim -a*" (esplicitamente o 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).

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>

History

È possibile elencare tutte le operazioni che l'utente ed il sistema hanno effettuato sul pool, tramite il seguente comando:

zpool history <pool>

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

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 e/o guasti hardware; tuttavia esse assolvono a questo 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). Questo meccanismo garantisce che i vecchi oggetti siano comunque conservati in caso di interruzione di corrente o crash fatale e che quindi al riavvio del sistema sul filesystem siano presenti o le nuove informazioni completamente valide, o le vecchie perfettamente integre. Soltanto dopo che i nuovi blocchi di dati

e metadati sono stati correttamente validati, che quelli vecchi vengono marcati come obsoleti (in modo da poter riutilizzare, in seguito, lo spazio che veniva occupato da essi). Viceversa, nei filesystem Journaled, in caso di perdita di alimentazione o crash del sistema, dati e metadati rischiano sempre di essere lasciati in uno stato incoerente (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 o perfettamente 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 performante 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, dagli impieghi SOHO (Small Office Home Office) fino ai grandi datacenter, per cui non vi è nessuna ragione valida per non utilizzarlo. È persino disponibile un porting per Windows[16] (ma non ho idea se esso sia sufficientemente stabile o meno).

Snapshot e cloni

Come abbiamo visto all'inizio, 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", viceversa, sono copie modificabili degli snapshot. All'inizio, essi non richiedono spazio aggiuntivo nel pool; quando vengono scritti nuovi dati sul clone allora 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. Ovviamente, 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.

Ritengo snapshot e cloni argomenti abbastanza avanzati (si possono fare danni con il ripristino degli snapshot), per cui rimando l'approfondimento alla documentazione ufficiale di OpenZFS.

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)[17]. Attualmente, ZFS è l'unico filesystem di classe enterprise sufficientemente maturo con strategia CoW, checksum dei dati e crittografia nativa (Bcachefs ha tali caratteristiche ma è ancora in fase embrionale, mentre Btrfs ha il CoW ed il checksum dei dati, ma non la crittografia nativa, ed in più soffre di alcuni problemi con

16 https://github.com/openzfsonwindows/openzfs/releases

17 https://blog.elcomsoft.com/2021/11/protecting-linux-and-nas-devices-luks-ecryptfs-and-native-zfs-encryption-compared/

il RAID5/6[18]). Non per nulla ZFS è la spina dorsale di moltissimi sistemi aziendali (inclusi grandi datacenter e supercomputer). La crittografia in ZFS può fare cose straordinarie come:

- assegnare chiavi di crittografia diverse ai vari dataset; ciò permette, ad esempio, che ogni utente abbia la sua HOME (o un'altra area) protetta con la propria 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 documentazione ufficiale, 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. ZFS non crittografa i metadati correlati alla struttura del pool (inclusi i nomi di dataset e snapshot), la gerarchia di dataset, le proprietà, le dimensioni dei file, i buchi nei file e le tabelle di deduplicazione (sebbene i dati deduplicati siano crittografati)[19]. 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 ≥ 0.8.4), ma può anche essere specificato esplicitamente tra quelli disponibili (tutti basati su AES). Attualmente, "aes-256-gcm" è il più robusto tra quelli supportati, ma anche quello con il maggiore impatto sulla CPU; nonostante ciò è ben tollerato anche su CPU di 10 anni fa. Le prestazioni sono paragonabili a quelle dei pool non crittografati, indipendentemente dalle dimensioni del file: da benchmark effettuati, si stima un impatto inferiore al 10% (che è anche minore di quello imposto da LUKS).

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[20] 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 la crittografia (come la compressione), avviene a livello di blocco: quando si attiva la crittografia su un dataset esistente, solo i blocchi scritti dopo tale abilitazione vengono crittografati; è pertanto fortemente consigliato 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 all'interruzione di

18 https://www.reddit.com/r/btrfs/comments/1bf35qx/will_the_write_hole_raid_5_6_bug_every_be_fixed/
19 https://www.reddit.com/r/zfs/comments/ienyuz/do_file_sizes_leak_with_zol_encryption/
20 https://iacr.org/archive/fse2002/23650264/23650264.pdf

alimentazione o a un crash di sistema, oppure viene ripristinato come se non fosse mai stato scritto. In ogni caso non c'è corruzione dei dati[21].

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>
```

IO Scheduler

ZFS ha il proprio scheduler di I/O, pertanto conviene forzare lo scheduler di I/O del kernel Linux a "none" per tutti i dischi completamente dedicati 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 SSD e NVMe.

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 su un laptop) non è consigliabile forzare lo scheduler di I/O di Linux a "none", in quanto ne risentirebbero negativamente i filesystem presenti sulle altre partizioni. 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 degrado delle performance di ZFS (che su SSD e NVMe è praticamente impercettibile).

ZFS su supporti rimovibili

ZFS non è attualmente resiliente nei confronti di errori di I/O, per cui non è adatto ad essere utilizzato su storage removibile (sebbene qualcuno, me compreso, lo usi) poiché falsi contatti sui

21 https://www.reddit.com/r/zfs/comments/1ad88wn/zfs_encryption_and_power_loss/

connettori USB possono provocare il blocco dei processi ZFS all'interno del kernel Linux, costringendo ad un riavvio del sistema (eventualmente tramite sequenza R.E.I.S.U.B) 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). Personalmente utilizzo ZFS da tanti 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 interno, con hardware di buona qualità.

Esempio: conversione MD/RAID1 in MIRROR ZFS

Eccoci finalmente ad un esempio concreto, il passaggio 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, metadati su unità a stato solido, ecc.

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: non avendo un disco "spare" di opportuna capacità qui si è proceduto alla copia eliminando prima uno dei dischi dall'array MD/RAID1. Ciò comporta la perdita della ridondanza, ed è assolutamente da evitare in caso di dati importanti (a meno che non si abbia un backup aggiornato).

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

SATA1 & SATA2 : SSD KINGSTON DC500M 960GB → ZFS special device
SATA3 & SATA4 : HDD WD GOLD 8TB → ZFS pool principale

Gli HDD WD da 8TB sono quelli in cui è presente l'array MD/RAID1 di partenza, mentre gli SSD KINGSTON sono aggiunti appositamente per il pool ZFS. Ecco i passi da seguire con attenzione:

- Ripristinare uno stato pulito del sistema con timeshift (se necessario), quindi installare i moduli ZFS per il kernel corrente ("linux*-zfs", "zfs-kmod" o altro pacchetto in base alla distro usata): "zfs-utils" (o "zfs") dovrebbe essere installato come dipendenza. Riavviare e accertare che "*zpool status*" funzioni senza errori (anche se non restituisce nulla).

- Commentare /mnt/STORAGE in /etc/fstab (come visto, con ZFS non serve).

- Dissociare /dev/sdc1 da MD/RAID1:

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

- Eliminare i metadati MD e tabella delle partizioni del disco rimosso dal RAID:

  wipefs -a /dev/sdc1 /dev/sdc

- Smontare /mnt/STORAGE e rimontare /dev/md127 in una posizione diversa (ad esempio /root/md127).

- Creare lo zpool “storage” completo di special device, specificando i percorsi sul bus SATA del WD rimosso da MD/RAID1 e di uno dei due KINGSTON:

  zpool create -m /mnt/STORAGE -o ashift=12
  storage /dev/disk/by-path/pci-0000:02:00.1-ata-3.0
  special /dev/disk/by-path/pci-0000:02:00.1-ata-1.0

- Verificare che il filesystem sia stato creato e montato:

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

- 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

- Verificare le proprietà appena impostate:

  zfs get all storage | egrep 'xattr|atime|compression|recordsize'

- Creare un dataset per ogni utente che deve poter memorizzare dati sul pool ZFS: avere dataset separati per ogni utente consente di applicare diverse politiche di snapshot, impostare lo spazio massimo che ciascun filesystem può utilizzare (quote), ottimizzare la dimensione dei record per dataset, allocare in modo diverso i dataset, ecc:

  zfs create storage/<user1>
  …
  zfs create storage/<last_user>

- Se necessario imporre delle quote (ripetere per ogni utente creato):

  zfs set quota=<size> storage/<user1>
  …

- Sul pool ZFS creare i dataset figli di ogni utente, imitando la struttura delle cartelle principali (top) presenti per ogni utente nell’array MD/RAID1; rispetto all’uso di un unico dataset per utente, ciò ha il vantaggio di poter applicare proprietà diverse ai diversi dataset (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 VIDEO BACKUP….; do zfs create storage/<user1>/$i; done
  …

  Volendo, si può anche procedere oltre nella gerarchia, per specializzare ulteriormente alcune proprietà dei dataset. Notare che le proprietà del dataset genitore vengono ereditate solo al momento della creazione dei figli (eventuali modifiche successive non sono propagate).

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

  chown -R <user1>:<user1> /mnt/STORAGE/<user1>
  …

- Stabilire quali dataset devono essere completamente allocati sullo special device, ad. es.:

  zfs set special_small_blocks=1M storage/<user1>/BACKUP_SYNC
  zfs set special_small_blocks=1M storage/<user1>/DOCUMENTI
  zfs set special_small_blocks=1M storage/<user1>/EMAIL_REPO
  …

- Verificare i dataset creati:

  zfs list

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

  mv /root/md127/<user1>/AUDIO/ /mnt/STORAGE/<user1>/AUDIO/*
  …

- Disassemblare completamente l'array MD/RAID1:

  mdadm --stop /dev/md127

- Eliminare i metadati MD e la tabella delle partizioni del secondo disco rimosso dal RAID:

  wipefs -a /dev/sdd1 /dev/sdd

- Creare il mirror per lo “special device” (aggiungendo il secondo SSD KINGSTON):

  zpool attach -o ashift=12 storage
  /dev/disk/by-path/pci-0000:02:00.1-ata-1.0
  /dev/disk/by-path/pci-0000:02:00.1-ata-2.0

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

  zpool attach -o ashift=12 storage
  /dev/disk/by-path/pci-0000:02:00.1-ata-3.0
  /dev/disk/by-path/pci-0000:02:00.1-ata-4.0

- Abilitare i servizi necessari affinché il pool ZFS venga riconosciuto e montato automaticamente al riavvio del sistema:

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

- Riavviare il PC, verificare che il pool ZFS sia stato riconosciuto e montato (con “zpool status”) ed effettuare il backup del sistema con timeshift (per il salvataggio delle modifiche).

Riferimenti

https://wiki.gentoo.org/wiki/ZFS
https://wiki.archlinux.org/title/ZFS
https://www.ixsystems.com/documentation/freenas/11.2/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://openzfs.github.io/openzfs-docs/Project%20and%20Community/FAQ.html
https://openzfs.github.io/openzfs-docs/Performance%20and%20Tuning/Workload%20Tuning.html
https://constantin.glez.de/2010/07/20/solaris-zfs-synchronous-writes-and-zil-explained/
https://www.truenas.com/community/threads/zil-and-l2arc-on-same-disk.23333/
https://blog.programster.org/zfs-record-size

Altri documenti utili

https://rudd-o.com/linux-and-free-software/ways-in-which-zfs-is-better-than-Btrfs
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

Pagine man principali

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

AVVERTENZE

Sebbene mi sia adoperato per far sì che le informazioni contenute in questo documento 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 qualsiasi danno, diretto, indiretto, incidentale e consequenziale legato all’uso, proprio o improprio delle informazioni contenute nel presente documento. Inoltre non sono responsabile dei contenuti e di eventuali errori o inesattezze presenti nei link a piè di pagina e nelle sezioni “Riferimenti” e “Altri documenti utili”.

Indice generale