Networking

ZFS: il file system del presente.†

Giuseppe Nantista,[a] Andrea Lora,[a] Augusto Pifferi,[a]

In questo rapporto tecnico analizzeremo il file system ZFS, formalmente Zettabyte File System, sviluppato a partire dal 2002 e annunciato in un paper da Jeff Bonwick et al.[1] della Sun Microsystems nel 2003. Descriveremo i principi che sono alla base della sua formulazione teorica e le sue caratteristiche, decantate dagli stessi autori, di semplicità di amministrazione, scalabilità e integrità del dato salvato. Infine citeremo alcuni aspetti pratici di questo file system nella quotidianità del suo utilizzo, sottolineando le implicazioni dietro ogni possibile scelta architetturale.
Keywords: ZFS, zettabyte, storage, file system.

1 Introduzione

Quando nel 2003 Jeff Bonwick e i suoi colleghi[1] della Sun Microsystems annunciarono la prossima uscita di un file system rivoluzionario sottolinearono da subito come ZFS, lo Zettabyte File System, presentasse caratteristiche di forte integrità dei dati, immensa capacità e facile amministrazione. Di fatto ZFS ha rappresentato una netta svolta nella concezione di un file system di tipo general purpose, spazzando via una serie di credenze e costrizioni che erano (e ahimè talvolta sono ancora oggi) di comune convinzione nell'ambito dei file system ad uso locale.

Sinteticamente i 3 punti sopra citati vengono ottenuti nel seguente modo:

- forte integrità dei dati tramite meccanismi di checksum di ogni singolo dato scritto su disco;
- immensa capacità tramite uno spazio di indirizzamento a 128 bit;
- facile amministrazione tramite l'uso di un metalinguaggio che permetta di esprimere in maniera concisa cosa si vuole fare.

ZFS introduce inoltre alcuni concetti importanti, che sono quelli di pooled storage, modello transazionale copy-on-write e di checksum auto validanti, ottenendo, a volte indirettamente, numerosi vantaggi come la possibilità di creare snapshot del file system in maniera istantanea o la possibilità di non usare più i meccanismi di file system check (il comando fsck ben noto e temuto da qualunque sistemista unix che usi ext come file system) o ancora la possibilità di correggere un errore su file system all'atto della lettura del dato errato, oltre che a richiesta sull'intero file system tramite il comando di scrub.

2 Limiti dei vecchi file system

Nel concepire i vecchi file system si faceva riferimento ad assunzioni che risultano oggi obsolete, come il fatto che proteggere i dati con l'uso di un sistema di checksum sia troppo oneroso per un computer. Questo non è più vero grazie alle capacità di calcolo sempre maggiori dei computer moderni, ma anche se così fosse i dati sarebbero troppo preziosi per non essere protetti con un algoritmo di checksum anche "light"; di base ZFS usa l'algoritmo di Fletcher[2] a 64 bit, un algoritmo di checksum che si propone come ottimo rapporto fra efficacia ed efficienza, molto più semplice dal punto di vista computazionale di un codice CRC, ma più debole per quanto riguarda la sua capacità di individuare errori. Per capire quanto questo sia importante si faccia riferimento a uno studio approfondito condotto dal CERN[3] che ha evidenziato come l'incidenza di errori "silenti" sia distruttiva nei file system tradizionali, incoraggiando un utilizzo diffuso dei meccanismi di checksum.

Un altro limite dei file system tradizionali è la corrispondenza fra un file system e un "volume". Facciamo un passo indietro, inizialmente un file system era correlato a un singolo disco. Questo approccio molto limitativo fu superato adottando il concetto di volume, che di fatto è un dispositivo a blocchi virtuale, ottenuto con differenti metodi a partire da dispositivi fisici, ad esempio dal

[a] *Istituto di Cristallografia, C.N.R. via Salaria km 29.300, 00015 Monterotondo, Italy*

† Rapporto tecnico 2014/11 con protocollo CNR-IC n. 1383 del 01/08/2014

mirror di due dischi o dalla loro concatenazione o da un RAID5, meccanismo che usa n dispositivi fisici di uguale capacità per ottenere un dispositivo virtuale con capacità pari a quella di n-1 dischi. Rimanendo sull'argomento il meccanismo RAID5 prevede che il controller dei dischi abbia una cache di memoria non volatile (NVRAM), per tamponare eventuali mancanze di energia elettrica durante una scrittura a disco, evento che comprometterebbe il contenuto dell'ultimo blocco di informazioni che si stava scrivendo al momento del blackout.

Un ulteriore limite è relativo alla massima dimensione in byte del file system. Anche usando 64 bit per indirizzare i blocchi di un file system, così come fatto dal alcuni recenti sistemi (stiamo di proposito ignorando i file system con spazio di indirizzamento a 32 bit che sono già ad oggi insufficienti), la massima dimensione allocabile sarebbe di 16 exabyte, un valore non così difficilmente raggiungibile nei prossimi 10 o 20 anni. ZFS ha optato quindi per un indirizzamento a 128 bit, che rende di fatto irraggiungibile la dimensione massima di un file system.[4]

3 Principi di ZFS

Gli studi che sono alla base dello sviluppo di ZFS si posano su pilastri fondamentali, evidenziati da Bonwick e colleghi già in fase di design.

3.1 Amministrazione semplice

La sintassi usata da ZFS è stata studiata per rendere immediato e sicuro impartire un qualunque comando al sistema, così creare un pool, un file system, uno snapshot o un clone sono operazioni che possono essere eseguite con massima facilità. Anche se tutte le operazioni compiute a livello di file system sono in carico agli amministratori di sistema e sono eseguite raramente, non c'è motivo per non semplificarle riducendo al minimo la possibilità, in caso di emergenza, di commettere errori irrecuperabili.

3.2 Pooled storage

Il concetto qui descritto è forse uno dei più rivoluzionari introdotti da ZFS. Consiste nella possibilità di mettere a fattore comune tutti i dischi fisici a disposizione a formare un'unica risorsa da cui pescare spazio per creare i file system. In questa maniera non ci sono problemi nel far crescere lo spazio a disposizione di uno di essi, nel senso che di default ogni file system ha a disposizione tutto lo spazio del pool su cui esso è stato creato. Nulla vieta tuttavia di impostare delle quote (lo spazio massimo occupabile da un FS) o delle reservation (lo spazio minimo garantito a un FS). È evidente come il concetto stesso di pooled storage si porta dietro quello appena descritto di file system di dimensione variabile.

Un'altra operazione consentita da un approccio del genere è il grow di un pool, ossia la possibilità di aumentare le sue dimensioni dinamicamente con la sola aggiunta di dischi. In fig. 1 è mostrata in maniera abbastanza esplicita una operazione di grow di un pool. Mostriamo da subito un caveat di questa operazione, mentre ci si aspetta che le proprietà di ridondanza vengano mantenute, in realtà si perde la capacità del sistema di resistere, senza perdita di dati, al guasto di un disco. Questo per via di una precisazione che è da fare relativamente all'operazione cosiddetta di restripe, l'operazione con cui il contenuto di un pooled storage viene ridistribuito sulla totalità dei dischi che compongono il pool. Questa operazione non viene compiuta automaticamente, pertanto il contenuto del pool non viene riarrangiato per occupare anche il disco appena aggiunto. In certi casi questo è possibile con dei workaround, come ad esempio esportare e reimportare un file system su un altro pool locale. Nella figura è stato anche introdotto un concetto che è quello del virtual device (vdev), una struttura intermedia che di fatto assume il significato di logical volume manager. Un

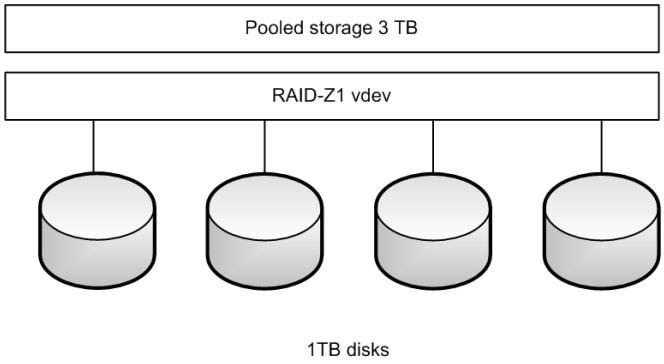

Fig. 1 Grow di un pool mediante aggiunta di un disco: situazione iniziale.

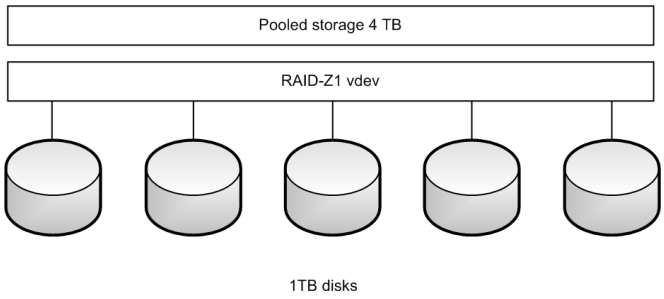

Fig. 2 Grow di un pool mediante aggiunta di un disco: soluzione attesa

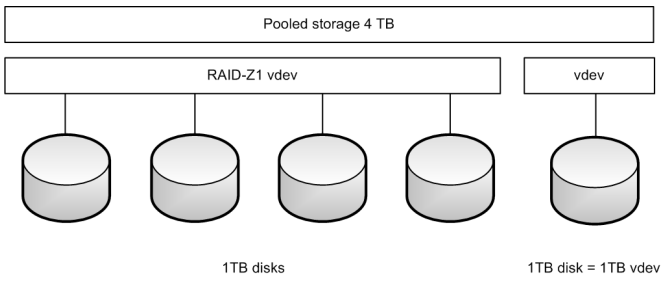

Fig. 3 Grow di un pool mediante aggiunta di un disco: soluzione ottenuta

vdev può consistere di uno o più dischi legati secondo una logica, come il mirror, la concatenazione o il RAID per creare lo spazio del pool.

3.3 Consistenza dei dati su disco

Al fine di evitare situazioni di inconsistenza dei dati salvati su disco è fondamentale che il sistema transiti da uno stato consistente a un altro. Un modo efficace è quello di applicare la politica Copy On Write, che consiste nel copiare un blocco ogni qual volta si renda necessario modificarlo, aggiornando i puntatori ad esso solo quando è stata completata l'operazione di scrittura. L'approccio COW apre due scenari interessanti, la creazione degli snapshot e dei cloni.

Concettualmente sono due oggetti molto simili, uno snapshot è un'istantanea del file system al momento della creazione dello stesso. Ogni qual volta un blocco viene modificato il COW crea un blocco nuovo e aggiorna i puntamenti ad esso (i puntamenti sono riferimenti a quel blocco, di fatto sono quindi dati scritti su un altro blocco seguendo una struttura gerarchica). La differenza fra creare uno snapshot e modificare un blocco è che l'area usata dai blocchi modificati non viene liberata, ma rimane referenziata per una futura operazione di rollback alla situazione precedente. Per loro natura gli snapshot sono read only, perché una modifica pregiudicherebbe l'operazione di rollback. Invece sono in tutto e per tutto file system in lettura/scrittura i cloni, ottenuti a partire dal file system parent che vivono di vita propria.

Un'altra forma di consistenza è quella relativa alla correttezza dei dati scritti. ZFS arricchisce l'area dei metadati associati a un file con il campo checksum, ossia un codice di parità relativo al blocco stesso, capace di rivelare la corruzione del blocco cui si riferisce, attivando i meccanismi di correzione dell'errore (healing) leggendo il blocco da un mirror. Questa operazione di verifica viene compiuta ogni volta che un file viene letto, ma l'amministratore può decidere di lanciare il comando di ***scrub***, che operando a bassa priorità legge tutto il file system alla ricerca di errori nascosti e li corregge. I metadati di checksum sono salvati nel blocco parent del blocco in oggetto, pertanto allocando i blocchi sul disco in maniera sparsa si aumentano le probabilità che in caso di corruzione di un blocco il suo blocco parent sia corretto.

3.4 Immensa capacità

Come già discusso brevemente nell'introduzione ZFS consente la creazione di file system di capacità praticamente non raggiungibile. Un singolo file o un file system possono raggiungere la dimensione massima di 16 exabyte (1 EB = 2^{60} byte), una directory può contenere 2^{48} file, uno zpool può comprendere 2^{64} vdev e contenere 2^{64} file system.

I file vengono collocati all'interno di blocchi, ZFS consente di utilizzare blocchi di dimensione variabile da 512 a 128 Kbyte per ridurre gli sprechi legati alla operazione di fit di un file in N blocchi.

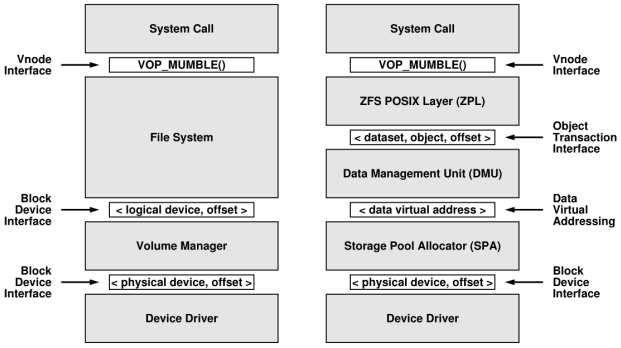

Fig. 4 Il modello di storage di ZFS

Prevede inoltre meccanismi di compressione del dato, cosicché lo spazio occupato su un file system sia ulteriormente ridotto a scapito della capacità computazionale impiegata per comprimere ed espandere i dati. Sono possibili diversi algoritmi di compressione, inclusi quelli della famiglia gzip, ma di default quello usato è l'algoritmo LZJB, sviluppato appositamente da Jeff Bonwich partendo dal nucleo originale di Lempel e Ziv (di qui l'acronimo).

Infine il meccanismo di deduplica fa sì che due blocchi uguali occupino lo spazio di uno solo.

4 Caratteristiche di ZFS

4.1 Il modello di storage di ZFS

Il modello di storage adottato da ZFS è mostrato in fig. 4, dove viene confrontato con il modello standard degli altri file system. I file system tradizionali accedono a un block device, sia esso un disco fisico o un volume logico ed esportano una interfaccia di tipo vnode al sistema operativo, ZFS introduce invece un approccio differente di seguito descritto:

1. *Lo storage pool allocator (SPA)*

   Il livello più basso dell'architettura ZFS è lo storage pool allocator, che interfacciandosi con unità fisiche, i dischi, tramite un accesso a blocchi, fornisce allo strato superiore dell'architettura un'interfaccia per allocare spazio disco. Poiché accede ai blocchi fisici dei dischi è lo SPA a prendere in carico l'operazione di checksum su di essi, memorizzando l'informazione nel blocco parent di quello scritto. Se in fase di lettura lo SPA si accorge di un mismatch fra i dati letti e il loro checksum effettua la correzione leggendo l'informazione dai blocchi corretti.

   Tutte le operazioni che in un file system tradizionali sono affidate al volume manager vengono in ZFS eseguite dallo SPA, che realizza mirror logici, concatenazioni, ecc. fra dischi appartenenti allo stes-

so pool. Quando lo SPA esegue queste operazioni crea un vdev, un dispositivo virtuale ottenuto secondo le logiche di ridondanza desiderate a partire da dispositivi fisici e/o altri vdev.

La logica con cui lo SPA alloca i blocchi dati partendo dai vdev è di tipo round-robin, effettuato in maniera dinamica per default. Se un dispositivo è stato recentemente aggiunto a un pool le scritture convergeranno prevalentemente su di esso fino a pareggiare l'occupazione percentuale del vdev stesso.

2. *La data management unit (DMU)*

   La DMU è la porzione dell'architettura ZFS che accedendo alla struttura di "virtual addressed blocks" offerta dallo SPA offre allo strato superiore, lo ZPL, un'interfaccia transazionale a oggetti. Questi oggetti sono unità di memorizzazione dati, identificati da un indirizzo a 64 bit, contenenti un massimo di 264 byte di dati, che lo ZPL può creare, distruggere, leggere e scrivere.

   La DMU implementa la logica Copy On Write, occupandosi di gestire il puntamento ai blocchi secondo una logica padre-figlio, realizzando una struttura gerarchica in cui il blocco padre di tutta la struttura è il cosiddetto überblock, un blocco ovviamente trattato in maniera particolare dalla DMU, che ne conserva una copia di backup da ripristinare in caso di transazione non eseguita correttamente per colpa ad esempio di un blackout. L'insieme di scritture da eseguire in ogni transazione è variabile, perché la DMU le raggruppa insieme, così da limitare il numero di accessi in scrittura ai blocchi incluso l'überblock.

3. *ZPL, ZFS Posix Layer*

   Al sistema operativo viene offerta una interfaccia di tipo vnode tradizionale, secondo le direttive POSIX (Portable Operating System Interface for uniX), che di fatto rappresenta l'unico compromesso architetturale di ZFS.

4.2 RAID-Z

Oltre ai meccanismi classici di mirror, concatenazione e stripe ZFS supporta anche il RAID-Z, nelle varianti Z1, Z2 e Z3, dove la cifra dopo la Z indica il numero di dispositivi guasti a cui il sistema resiste senza perdita del dato. Il modello a cui si ispira il RAID-Z è lo stesso del RAID-5, che permetteva di usare n dischi di uguale dimensione per ottenere un volume di capacità pari a quella di n-1 dischi. Analogamente RAID-Zi usando n vdev di uguale dimensione ottiene uno storage pool di capacità pari a quella di n-i. È possibile anche usare vdev di dimensione differente perdendo parte della capacità dei dischi più grandi.

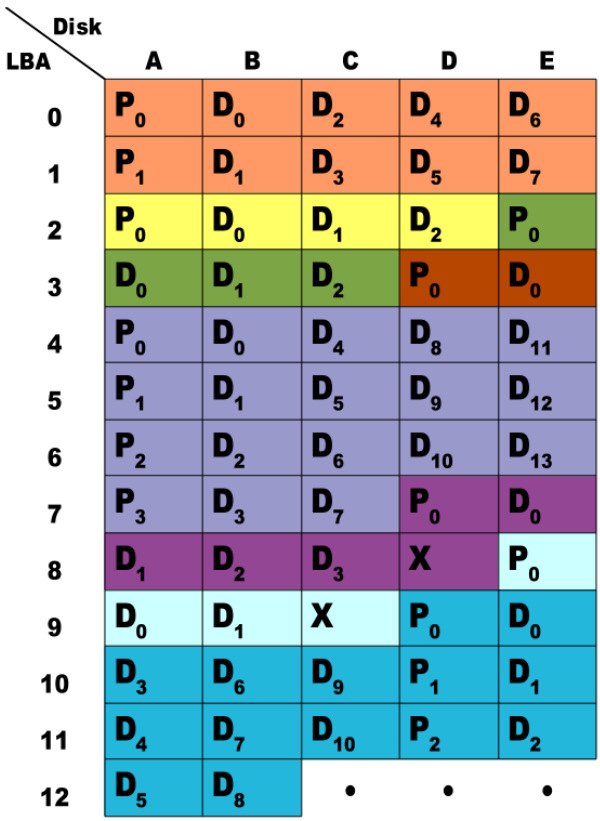

Fig. 5 Distribuzione dei blocchi in RAID-Z

Il punto in cui RAID-Z si differenzia maggiormente dai suoi predecessori è la logica di distribuzione dei blocchi di parità, quelli cioè che consentono di recuperare l'informazione in caso di guasto. Il tradizionale RAID-5 effettua lo XOR (OR esclusivo) sui blocchi analoghi dei primi n-1 dischi e scrive il risultato nell'ultimo blocco, consentendo il recupero per differenza in caso di guasto, ma non in caso di errore di tipo "silent", situazione in cui il sistema rischia di correggere dati già corretti a partire da dati di parità inconsistenti. Un problema del tutto analogo si manifesta con il cosiddetto "write hole", la situazione cioè in cui fra la scrittura del dato e la scrittura della parità avviene un blackout; il modo comunemente utilizzato per resistere al write hole è l'uso di memorie NVRAM che mantengono il dato anche in caso di interruzione della corrente elettrica.

ZFS invece garantisce la coerenza dei dati grazie a sopra citato meccanismo "copy-on-write" che assicura l'atomicità di ogni singola scrittura, in aggiunta a questo il RAID-Z arricchisce il pool dei dati di informazioni di ridondanza che sono utili in caso di danneggiamento totale o parziale di un disco. Non è possibile come in RAID5 correggere dati validi usandone di inconsistenti, perché ZFS sa sempre quale dato è corretto e quale no grazie ai checksum.

Lo schema di RAID-Z è simile a quello di RAID5 nel senso che utilizza sempre un approccio di tipo dati + pa-

rità, la differenza sta nel fatto che ogni blocco che viene scritto ha una dimensione variabile, pertanto anche lo stripe, ossia l'insieme di blocchi fisici sui dischi, su cui una scrittura viene distribuita, ha una dimensione variabile. In fig. 5 viene illustrato il metodo di scrittura dei blocchi dati (evidenziati da una lettera D) e dei blocchi di parità (lettera P).

L'esempio in questione riporta il caso di 5 dischi (lettere A-E in ascissa) scritti con algoritmo di parità RAID-Z1, mentre in ordinata sono riportati i blocchi logici dei dischi (logical block addresses), ogni stripe di un colore differente rappresenta un blocco ZFS.

Nell'esempio sono stati anche riportati 2 errori localizzati, evidenziati con una lettera X, corrispondenti a un fallimento nella verifica del checksum, tali errori sono recuperabili tramite i dati corretti e le parità. Il massimo guasto tollerato da questo sistema è pari a un intero disco (una colonna).

4.3 Ditto Blocks

Quando un errore compromette la possibilità di leggere un blocco i dati in esso contenuti sono persi, a meno di meccanismi di ridondanza come quelli appena descritti. Ma mentre la perdita di un blocco dati ha conseguenze limitate, la perdita di un blocco intermedio nella struttura gerarchica del file system ZFS, contenente riferimenti e metadati, comporta l'impossibilità di accedere a tutti i blocchi da esso dipendenti.

Per salvaguardare l'integrità di questi blocchi, che da uno studio compiuto al riguardo occupano circa il 2% dello spazio disco, è possibile crearne delle copie in altri settori del disco, così da minimizzare la possibilità che un singolo guasto possa cancellare entrambe le copie. Se il sistema è munito di un solo disco, la distanza minima a cui tenere il blocco di copia è pari a 1/8 del disco stesso. Più è importante il blocco, ossia più in alto esso si trova nella struttura gerarchica, più copie possono esserne fatte per salvaguardarlo; ZFS di default assegna due locazioni (DVA, Data Virtual Address) per i blocchi intermedi e tre per i dati di accesso globale.

Nella terminologia ZFS vengono chiamati Ditto Blocks i blocchi in cui memorizzare le copie di ridondanza dei DVA.

5 Caratteristiche avanzate e precisazioni

Discutiamo in questo paragrafo alcuni aspetti che è importante conoscere prima di operare con il file system ZFS. Facciamo esplicitamente riferimento al documento di Andrew Galloway[5] del 2011, reperibile solo in rete.

Partiamo da una affermazione spiritosa: Il fatto che tu possa fare una cosa non implica che tu debba. Questo può far sorridere, ma effettivamente esprime bene il concetto per cui alcune soluzioni consentite da ZFS non rappresentano l'ottimo e vanno pertanto considerate con i loro pro e i loro contro. ZFS impone pochissimi limiti, ma il giusto compromesso non si ottiene mai spingendosi al massimo consentito dal sistema. Pensiamo ad esempio al fatto che si possano raggruppare 2^{64} vdev in un pool, questo è evidentemente un valore cui non bisogna nemmeno lontanamente avvicinarsi, pena la crescita smisurata del numero di operazioni di I/O al secondo (iops).

5.1 La deduplica

Questa tecnica consente di risparmiare grosse quantità di spazio su disco a discapito di un forte utilizzo di RAM per memorizzare le informazioni necessarie a sfruttarla e di CPU per valutare se un blocco è deduplicabile. Ogni volta che un blocco viene consegnato dallo ZPL alla DMU, questa ne calcola l'hash tramite algoritmo SHA256 e verifica se il blocco in questione può essere deduplicato, ossia se esiste già un blocco uguale (più precisamente un blocco con lo stesso hash, quindi attenzione perché c'è una seppur remota probabilità di collisione) scritto sul disco, nel qual caso non lo riscrive ma salva un riferimento al blocco già presente. Per funzionare il meccanismo fa uso di tavole di deduplica, che vengono tenute in RAM e vengono create e consultate in fase di scrittura e lettura dai file system. Questo può comportare un grosso dispendio di memoria, ben oltre il vantaggio di aver risparmiato spazio su disco. Le indicazioni di massima da parte di SUN sono quelle di compiere due valutazioni: prima tramite il tool zdb (ZFS debugger) valutare la possibile deduplication ratio, ossia il fattore di risparmio che si otterrebbe attivando la deduplica, se questo valore supera il fattore 2 allora ci può essere convenienza; la seconda verifica va compiuta sulla quantità di memoria necessaria ad attivare la deduplica a fronte di quella disponibile. La memoria richiesta si calcola moltiplicando il numero di blocchi allocati per 320, che è il valore in byte di una entry nella DDT. In appendice, in tab. 1, mostriamo un possibile scenario in cui il fattore di guadagno derivato dalla deduplica è nullo. In questo caso attivare la deduplica comporterebbe un consumo di RAM pari a 326,4 MB di memoria, pari a 320 byte x 1,02M (numero di blocchi in uso).

5.2 La compressione

Non ci sono dubbi a riguardo, la compressione va usata. Il guadagno in termini di I/O ottenuto è notevole e la perdita in termini di risorse CPU è trascurabile, specie se la compressione utilizzata è la LZJB, la versione di Jeff Bonwich della famiglia di algoritmi di compressione Lempel Ziv, più leggero e di conseguenza meno oneroso per il processore. È tuttavia possibile utilizzare l'algoritmo gzip selezionando il flag di "compression level" al fine di trovare il giusto rapporto fra costo computazionale e risparmio di spazio.

5.3 L'uso di memoria

ZFS usa tutta la memoria che gli viene messa a disposizione. Una feature di ZFS molto utile per l'aumento delle prestazioni è la ARC, Adaptive Replacement Cache, un meccanismo molto avanzato di caching dei dati. In aggiunta è possibile usare la L2ARC, che fa uso di unità disco molto veloci (ad esempio dischi SSD) per avere una quantità superiore di memoria cache, seppur meno prestante della RAM del sistema.

5.4 Amministrazione semplice ma "not for dummies"

I comandi di ZFS sono immediati, ma è necessario sapere cosa si sta facendo prima di lanciare un comando. Questo ci riconduce alla prima precisazione di questo paragrafo, il fatto che tu possa fare una cosa non implica che tu debba.

Vogliamo fare riferimento all'operazione, riportata al paragrafo 3 e in fig. 1, di grow di un pool; se fosse possibile realizzarla così come auspicato l'operazione manterrebbe il livello di affidabilità iniziale, ossia la resistenza del sistema al guasto di uno qualunque dei dischi fisici; invece quello che succede, e che viene scherzosamente definito "hating your data", è che i dati scritti dopo l'operazione di grow saranno distribuiti, in funzione dello spazio disponibile, su due vdev, uno con affidabilità di un disco guasto su quattro e uno senza alcun livello di affidabilità, con conseguente perdita di tutti i dati in esso contenuti in caso di guasto fisico. Un modo per evitare questo è quello di creare a parte un nuovo pool, esportare i file system, distruggere la struttura precedentemente creata e ricrearla da capo con l'aggiunta del nuovo disco.

Ci sono poi particolari features del sistema che vanno apprese pena il fallimento degli intenti. Per esempio nelle operazioni send/receive incrementale del file system è importante che il filesystem ricevente non sia modificato. Eseguire una semplice visualizzazione di un file, che non portano modifiche ad esso, sono però sufficienti a cambiare il contenuto del filesystem, poiché sono cambiati, in questo esempio, i metadati di accesso. La risoluzione del problema qui e' settare il parametro readonly nel filesystem, che ne impedisce qualsiasi modifica.

5.5 ZIL: ZFS Intent Log

Una discussione estesa di cosa sia ZIL è fuori dallo scopo di questo documento. Lo scopo dello ZIL è quello di proteggere i dati in caso di fallimento della macchina. Per questo potremmo assimilarlo ad un sistema di journaling di un file system, ma come ha fatto notare Toponce[6] è più simile ad RDBMS che rispetti le specifiche ACID. Qualunque scrittura sincrona richiesta a ZFS verrà scritta sullo ZIL, e solo dopo l'ACK di questo verrà mandata la conferma allo strato applicativo. ZFS poi eseguirà il commit secondo le sue transazioni sullo strato fisico. In una situazione in cui non venga specificato un device apposito, lo ZIL corrisponde ad una zona del pool corrente, con gli stessi tempi di accesso. L'esistenza o la non esistenza dello ZIL non cambia il funzionamento e le performance del sistema. ZFS permette di inserire nel pool uno SLOG (Separate Log Device) che sarà la nuova locazione dello ZIL. In caso di scrittura sincrona quindi ZFS scriverà sullo SLOG i dati, ricevuto l'ACK dallo SLOG provvederà a trasmetterlo allo strato applicativo, nel frattempo potrà scrivere i dati sui suoi dischi. Se il dispositivo che funge da SLOG è un dispositivo ad alte performance (SSD o RAM con backup a batteria) tutte le scritture sincrone beneficeranno di tempi di esecuzione ben inferiori a quelli che avrebbero ottenuto in una configurazione senza SLOG. È impossibile stimare i miglioramenti ottenuti da uno SLOG, visto che cambiano in base al carico di lavoro, ma se ZFS è utilizzato per scritture sincrone (un server DB ne è un esempio) il suo utilizzo è fortemente consigliato. Secondo il workflow sopra illustrato se ci fosse un problema alla macchina tra quando lo SLOG ha inviato l'ACK ma lo strato fisico ancora non ha terminato il commit lo SLOG è il solo contenitore di informazioni che lo strato applicativo dà per garantite. E' quindi norma comune configurare lo SLOG come mirror di due dischi, poiché la rottura del componente può generare perdita di dati.

6 Licenze d'uso

Una annosa questione relativa a ZFS è il suo mancato inserimento all'interno del kernel di Linux, per via di un conflitto fra le licenze sotto cui i due prodotti open source vengono rilasciati.

ZFS è licenziato secondo la CDDL, Common Development and Distribution License, che impedisce esplicitamente una redistribuzione del software secondo altre condizioni, mentre il kernel di Linux è distribuito sotto GNU GPLv2, General Public License versione 2, che impone che un software che venga distribuito insieme ad esso debba essere licenziato GPL, anche se giudicato separato e indipendente. Un siffatto comportamento viene spesso indicato come "virale" e impedisce che i due progetti possano essere distribuiti congiuntamente.

Questo aspetto è stato trattato più approfonditamente in altre sedi[7], in questo rapporto tecnico si vuole solo dare un'idea sommaria del perché un prodotto così ben strutturato e open non è stato integrato in quello che è il sistema operativo open source per antonomasia.

7 Conclusioni

Il file system ZFS è stato sviluppato ormai oltre un decennio fa e rappresenta ad oggi il miglior file system per sistemi stand alone. Le sue caratteristiche di affidabilità, scalabilità e semplicità di amministrazione lo rendono ineguagliabile al confronto con i sistemi analoghi usati in ambito personal computer e server, tuttavia una difficoltà

di carattere legislativo ne ha impedito lo sviluppo diffuso. Installare ZFS in ambiente linux è possibile, distribuire i binari di installazione in un sistema con kernel licenziato GPL no. Il progetto ZFS on Linux ornisce, nei limiti del possibile tutto il supporto necessario per adottare questo file system in ambienti open source tradizionali.

Perché "il file system del presente" e non del futuro? La risposta è semplice, il futuro dei sistemi informatici vede sempre più affermarsi le architetture virtualizzate e lo storage distribuito dei dati, con strutture scalabili orizzontalmente all'infinito senza i limiti fisici dello chassis in cui risiede un server. ZFS non è un file system distribuito, anche se dal suo sviluppo hanno visto la luce progetti paralleli, come Lustre, che si pongono come obiettivo quello di applicare ai sistemi distribuiti le valutazioni compiute durante lo studio e lo sviluppo portati avanti da Bonwick e colleghi.

Il nostro gruppo di lavoro ha utilizzato ZFS come file system di storage per altri sistemi realizzati, come ad esempio Mercurio, il server di posta elettronica dell'Area della Ricerca RM1 del CNR o Pandora, la piattaforma di storage in the cloud.

Per il server di mail sono state utilizzate pesantemente le funzionalità di snapshot di ZFS. Affiancando alle feature uno script di automatizzazione è stato possibile realizzare degli snapshot incrementali e granulari: vengono creati gli snapshot ogni quarto d'ora (con retention degli ultimi 4), ogni ora (con retention 24), ogni giorno (con retention 30), ogni mese (retention 12) e ogni anno (retention illimitata). Lo spazio effettivamente consumato dai backup aumenta al crescere delle email cancellate dagli utenti, visto che le mail che attualmente sono presenti nelle caselle non pesano sul backup. Lo stesso approccio senza l'uso di ZFS avrebbe richiesto spazi di storage decisamente superiori.

Per eseguire il backup dei pool ci affidiamo alle capabilities di send/receive di ZFS, che ci permettono di eseguire il salvataggio dei dati delle piattaforme in produzione senza nessuna interruzione. Abbiamo attivato la compressione sul pool del ricevente, visto che gli attributi possono essere settati indipendentemente, beneficiando di un risparmio di spazio.

ZFS offre anche il supporto allo ZVOL, un dispositivo a blocchi creato sopra il pool ZFS, che poi può essere esportato tramite un protocollo che gestisca questo tipo di device, come ad esempio iSCSI. Questo consente di utilizzare ZFS come backend disco di sistemi di virtualizzazione o di usarlo come block device di un sistema operativo sempre tramite iSCSI.

8 Appendice

Riportiamo di seguito alcuni comandi previsti dalla sintassi ZFS, così da mostrare praticamente uno dei punti fondamentali del file system, la sua semplicità di amministrazione. Come creare uno storage pool denominato "tank" usando come vdev il mirror dei due dischi fisici c2d0 e c3d0

```
# zpool create tank mirror c2d0 c3d0
```

Come creare un file system denominato "home" come figlio di "tank" ed esportarlo al mountpoint /export/home

```
# zfs create tank/home
# zfs set mountpoint=/export/home tank/home
```

Come creare le home directory per alcuni utenti. Per l'ereditarietà della operazione di export queste saranno automaticamente esportate ai mountpoint /export/home/<nome>

```
# zfs create tank/home/giuseppe
# zfs create tank/home/andrea
# zfs create tank/home/augusto
```

Come aggiungere al pool altro spazio, un vdev ottenuto dal mirror di c4d0 e c5d0

```
# zpool add tank mirror c4d0 c5d0
```

Come esportare automaticamente tutte le home directories via NFS

```
# zfs set sharenfs=rw tank/home
```

Attivare la compressione sul pool "tank"

```
# zfs set compression=on tank
```

Limitare la quota di Giuseppe a 10 gigabyte

```
# zfs set quota=10g tank/home/giuseppe
```

Garantire ad Andrea una reservation di 20 gigabyte

```
# zfs set reservation=20g tank/home/andrea
```

Creare uno snapshot della home directory di Augusto

```
# zfs snapshot tank/home/augusto@tuesday
```

Effettuare l'operazione di roll back a uno snapshot precedente

```
# zfs rollback tank/home/augusto@monday
```

Accedere in lettura a una versione precedente di un singolo file. Wednesday è il nome dello snapshot.

```
$ cat ~maybee/.zfs/snapshot/wednesday/foo.c
```

```
# zdb -S tank
Simulated DDT histogram:

bucket              allocated                       referenced
______   ______________________________   ______________________________
refcnt   blocks   LSIZE   PSIZE   DSIZE   blocks   LSIZE   PSIZE   DSIZE
------   ------   -----   -----   -----   ------   -----   -----   -----
     1    1.00M    126G    126G    126G    1.00M    126G    126G    126G
     2    11.8K    573M    573M    573M    23.9K   1.12G   1.12G   1.12G
     4      370    418K    418K    418K    1.79K   1.93M   1.93M   1.93M
     8      127    194K    194K    194K    1.25K   2.39M   2.39M   2.39M
    16       43   22.5K   22.5K   22.5K      879    456K    456K    456K
    32       12      6K      6K      6K      515    258K    258K    258K
    64        4      2K      2K      2K      318    159K    159K    159K
   128        1     512     512     512      200    100K    100K    100K
 Total    1.02M    127G    127G    127G    1.03M    127G    127G    127G

dedup = 1.00, compress = 1.00, copies = 1.00, dedup * compress / copies = 1.0

DDT memory needed = 326,4 MB
```

Tab. 1 Verifica della possibilità di deduplica del file system.

Riferimenti

1 J. Bonwick, M. Ahrens, V. Henson, M. Maybee, M. Shellenbaum, The Zettabyte File System (2003).

2 J. Fletcher, An arithmetic checksum for serial transmissions, IEEE Transactions on Communications 30 (1) (1982) 247–252. doi:10.1109/TCOM.1982.1095369.

3 B. Panzer-Steindel, CERN/IT, Data integrity, Draft 1.3 8. (April 2007).

4 S. Lloyd, Ultimate physical limits to computation, Nature 406 (2000) 1047–1054.

5 A. Galloway, Things about ZFS that nobody told you (2011).

6 A. Toponce, ZFS Administration, Appendix A - Visualizing the ZFS Intent Log (2013).

7 R. Williams, R. Townsend, (Attorneys at Law), ZFS on Linux: Copyright and Licensing Issues.