POLITECNICO DI TORINO

Corso di Laurea Magistrale
in Ingegneria Informatica
(Computer Engineering)

Tesi di Laurea Magistrale

Sistemi di storage distribuito su Kubernetes

Politecnico
di Torino

CRITICALCASE

Relatore
prof. Fulvio Risso
Tutor aziendale
Pasquale Lepera

Candidato
Salvatore Russo

Anno Accademico 2020-2021

Alla mia famiglia

Sommario

Il continuo aumento delle applicazioni a microservizi stateful e la crescita del multi- cloud hanno contribuito alla necessità di trovare un'astrazione dello storage che permetta di garantire determinate proprietà svincolandosi sempre di più dall'hardware sottostante, sfruttando le infrastrutture iperconvergenti e la potenza di strumenti di orchestrazione come Kubernetes.

Il lavoro di tesi si pone come obiettivo l'analizzare le soluzioni di storage distribuito presenti sul mercato, ponendo maggiore attenzione sul mondo open source, per poi sceglierne una e misurarne le prestazioni, comparando eventuali differenze nel deploy e nel comportamento su infrastruttura on-premise e sul cloud pubblico.

Dallo studio delle architetture, dall'analisi del supporto da parte della community open source e del grado di interesse, le due soluzioni maggiormente apprezzate risultano essere MinIO e Ceph. A differenza di MinIO però, che gestisce solamente storage a oggetti, Ceph consente anche la creazione di storage a blocchi e filesystem. Per questa maggiore flessibilità, Ceph viene scelto come argomento di analisi, utilizzando Rook come storage orchestrator su Kubernetes.

Il deploy del cluster Rook-Ceph è stato eseguito su tre diverse infrastrutture: la prima soluzione analizzata prevede l'utilizzo di una SAN, la seconda prevede dei dischi direttamente connessi ai nodi, la terza invece ha visto la creazione di un cluster su Azure Kubernetes Service, preso come esempio di soluzione di cloud pubblico. Dopo aver evidenziato le differenze dei file di deploy di Rook-Ceph per i vari ambienti, sono stati creati due filesystem per compararne le performance: il primo filesystem prevede la replica dei dati, il secondo filesystem invece sfrutta la tecnologia Erasure Code, la quale prevede la suddivisione del dato in data chunk e coding chunk, garantendo così la disponibilità del dato utilizzando un'inferiore quantità di storage.

Utilizzando il tool open source Fio sono stati effettuati dei test in lettura e in scrittura cambiando la dimensione dei blocchi, e sono stati misurati i valori di IOPS, bandwidth, latenza, percentuale di occupazione di CPU e RAM. Tra gli aspetti più interessanti ottenuti dai test, risaltano le performance migliori ottenute in lettura da Rook-Ceph su AKS se comparate all'accesso diretto ai dischi Azure, e le migliori performance in scrittura ottenute su filesystem con erasure code nei confronti del filesystem replicato, sia su infrastruttura on-premise, sia su AKS.

Indice

Parte I

Introduzione

Capitolo 1

Introduzione

In un mondo che va sempre più digitalizzandosi, con le applicazioni che aumentano in termini di numero e complessità, nel campo aziendale aumenta la necessità di avere potenza di calcolo per poter processare diversi tipi di task.

Con il passare del tempo l'hardware in possesso delle aziende si è rivelato non più sufficiente e/o conveniente, di conseguenza è sorta la necessità di trovare soluzioni alternative: una di queste è il cloud.

1.1 Evoluzione del cloud: dall'on-premise al cloud pubblico

La nascita del cloud ha permesso a utenti e imprese di non dover più gestire dei propri server, liberandosi così dall'impegno della manutenzione dell'hardware e dell'aggiornamento di componenti e sistemi operativi, potendo così investire un maggior numero di risorse nella produttività e nello sviluppo dei propri applicativi.

Cloud privato, cloud pubblico e cloud ibrido sono soluzioni diverse nel mondo del cloud computing. Il cloud privato è la soluzione che più si avvicina all'avere un data center di proprietà: essendo un'architettura on-premise, quindi fisicamente collocata all'interno dell'organizzazione, offre maggiore controllo e sicurezza, insieme a maggiore personalizzazione e ai vantaggi del cloud come scalabilità ed elasticità, ma richiede di sostenere le spese di manutenzione, personale e gestione. Con il cloud pubblico invece l'infrastruttura è gestita da un fornitore esterno, il quale vende a più organizzazioni macchine virtuali o servizi. Poiché gli stessi server sono contemporaneamente utilizzati da più aziende, si parla di "multi-tenant". I vantaggi del cloud pubblico sono numerosi, ad esempio l'abbassamento dei costi , in quanto si paga solamente per le risorse effettivamente utilizzate e non bisogna più gestire i costi del personale e della manutenzione, e la garanzia di resilienza in quanto il fornitore può mettere a disposizione server dislocati in diverse aree del mondo, quindi non si hanno interruzioni del servizio in caso di problemi in un data center. Con il cloud ibrido invece un'organizzazione utilizza sia il proprio data center sia una soluzione pubblica come ad esempio Amazon Web Services, Azure, Google Cloud. Le ragioni possono

essere diverse: uno scenario può essere gestire i dati nei propri server per averne maggiore controllo e sfruttare la grande potenza di calcolo del cloud pubblico.

1.2 Virtualizzazione: Container e Kubernetes

L'utilizzo dei container negli ultimi anni ha permesso di rendere più semplice concentrare gli sforzi sul software piuttosto che sull'hardware. Il più grande effetto di questa evoluzione è lo svilupparsi dell'approccio a microservizi: invece di creare un'unica applicazione monolitica, si punta a scomporla in parti più piccole logicamente indipendenti che cooperano tra di loro.

I vantaggi di un' architettura a microservizi sono numerosi, ma si vuole porre l'attenzione sulle proprietà di scalabilità e agilità. Per quanto riguarda la scalabilità, la possibilità di eseguire ogni microservizio in un diverso container permette di creare nuove istanze quando l'applicazione richiede maggiore potenza computazionale, o allo stesso modo di ridurre le risorse occupate nel caso in cui il numero di richieste diminuisca. In particolare, il vantaggio consiste nella possilità di creare nuove istanze del solo microservizio in quell'istante maggiormente utilizzato, senza dover replicare l'intera applicazione.

Sono vari gli scenari in cui la proprietà di agilità si rivela un grande vantaggio: un esempio può essere trovato in fase di update dell'applicazione, in quanto è possibile creare un container con la nuova versione del microservizio e solo successivamente eliminare il container con la versione precedente. Così facendo, viene eseguito un update senza creare nessun downtime. Altro importante scenario è la possibilità di svincolare l'applicazione dall'hardware replicando un container in più nodi garantendo resilienza.

È a questo punto necessario un software che possa occuparsi in maniera intelligente, automatica ed efficiente della gestione dei container. Tale software prende il nome di orchestratore e Kubernetes è l'esempio di maggior successo.

Kubernetes è un sistema open source creato dagli ingegneri di Google in grado di gestire risorse su ambienti on-premise e su cloud pubblico. La gestione delle risorse consiste nell'automatizzare una serie di attività come, ad esempio, lo scheduling dei microservizi, la creazione di repliche, la reazione a failure hardware o software e l'auto-scaling.

1.3 Iperconvergenza e storage distribuito

Il mondo dei data-center ha visto un'evoluzione anche nell'infrastruttura: mentre prima i singoli componenti erano separati e individualmente gestiti, si è progressivamente puntato alla semplificazione e all'incremento dell'agilità. Questo obiettivo ha portato all'utilizzo di sistemi integrati, ovvero tutti i componenti (computazione, storage e networking) all'interno di un unico nodo. Lo sviluppo del software e la nascita della virtualizzazione hanno permesso un ulteriore step anche nella direzione della riduzione dei costi. La nascita delle infrastrutture iperconvergenti infatti si basa su un'interfaccia software che garantisce la gestione centralizzata di tutti gli elementi che compongono l'architettura. Tale astrazione permette l'utilizzo di hardware commodity, andando quindi ad abbassare i costi in quanto

non è necessario l'acquisto di hardware specifico per garantire compatibilità, e di centralizzare e di automatizzare il controllo dell'intero data-center, potendo anche aggiungere nuovi nodi successivamente.

L'adozione di infrastrutture iperconvergenti porta con sé anche un diverso approccio allo storage nel data-center. Se infatti non è più necessario adottare delle soluzioni ad hoc e costose, è adesso possibile utilizzare delle soluzioni standard sfruttando lo storage direttamente connesso ad ogni nuovo nodo che viene aggiunto, attraverso un overlay software che si occupa di gestirlo. In questo ambito sono state sviluppate numerose implementazioni, molte delle quali open source.

Questo elaborato di tesi si pone come obiettivo l'analisi di alcune di queste implementazioni per identificare una soluzione che insieme a Kubernetes offra una valida e completa astrazione degli strati sottostanti.

Capitolo 2

Nozioni necessarie

2.1 Storage distribuito

2.1.1 Formati di storage

Esistono tre diversi formati di storage:

- **file**: il sistema monta un virtual drive contenente file che però sono distribuiti su diverse macchine. I file sono gestiti attraverso una gerarchia. È una soluzione spesso usata nei NAS (Network Attached Storage) e nei DAS (Direct Attached Storage). Mentre scalabilità e prezzi sono i principali vantaggi, i tempi di accesso sono il principale svantaggio al crescere delle dimensioni.
- **blocchi**: i dati sono raccolti in volumi chiamati blocchi, permettendo di raggiungere prestazioni superiori utilizzando un software che si occupa di localizzare i blocchi attraverso l'indirizzo che gli viene assegnato. Un esempio di sistema distribuito di storage a blocchi sono le SAN (Storage Area Network).
- **oggetti**: ogni oggetto tipicamente include il dato in sé, i metadati che lo descrivono e un identificatore globale univoco. La suddivisione in oggetti permette di fornire funzionalità che gli altri modelli non hanno, come interfacce programmabili dalle applicazioni ecc. La presenza dell'ID univoco permette l'utilizzo di una struttura piatta, la quale rende marginale l'effettiva ubicazione dei file, migliorando la scalabilità ma peggiorando le performance, in quanto rende necessario ricostruire l'oggetto per intero in caso di modifica del file.

2.1.2 Vantaggi dello storage distribuito

I vantaggi nell'utilizzare soluzioni di storage distribuito sono numerosi, vengono di seguito evidenziati i principali.

- **scalabilità**: è possibile aggiungere più spazio di storage aggiungendo più nodi al cluster (scalabilità orizzontale).

- **ridondanza**: più copie dello stesso dato possono essere memorizzate garantendo disponibilità, backup e disaster recovery.

- **costi**: è il principale vantaggio del cloud, ovvero la possibilità di usare hardware commodity più economico per memorizzare grandi quantità di dati a costi inferiori.

- **performance**: i dati possono essere forniti in minor tempo accedendo a dei server più vicini all'utente finale o garantendo parallelizzazione per i file più grandi.

2.1.3 Principali proprietà

I sistemi di storage distribuito possono avere una o più delle seguenti proprietà:

- **partitioning**: è la caratteristica di poter distribuire i dati tra i nodi del cluster garantendone comunque l'accesso senza effetti negativi.

- **replication**: possibilità di duplicare lo stesso dato in più nodi diversi del cluster e di mantenerne la consistenza quando l'utente lo aggiorna.

- **fault tolerance**: possibilità di garantire la disponibilità del dato anche quando uno o più nodi del cluster presentino problemi.

- **elastic scalability**: possibilità di fornire all'utente più o meno spazio di archiviazione a seconda delle necessità e di permettere agli operatori di scalare il sistema modificando le unità di storage nel cluster.

Poiché per il teorema **CAP** (Consistency, Availability, Partition Tolerance) non è possibile garantire queste tre proprietà contemporaneamente, spesso si preferisce sacrificare la consistenza in favore della garanzia della sopravvivenza dei dati.

2.2 Kubernetes

2.2.1 Architettura di un cluster Kubernetes

Un cluster Kubernetes è costituito da nodi, i quali possono essere sia host fisici che virtuali e sono suddivisi in due categorie a seconda della loro funzione: i worker nodes si occupano di eseguire le applicazioni dell'utente e di inviare informazioni sul proprio stato ai master nodes, i quali hanno invece lo scopo di gestire il cluster occupandosi del control plane.

L'unità fondamentale in un cluster è il Pod. Esso contiene uno o più container ed è strettamente connesso a un nodo. Per garantire controllo sugli accessi e per avere una separazione logica tra le risorse vengono creati e utilizzati i namespace. Bisogna però sottolineare che pod appartenenti a namespace differenti possono comunque interagire tra di essi.

Solitamente il deploy dei Pod non viene effettuato in maniera diretta ma vengono utilizzati i controller:

- **ReplicaSet**: in fase di deploy viene specificato il numero richiesto di repliche del pod. In caso di failure, ad esempio un nodo non più disponibile, una nuova replica viene automaticamente avviata in uno degli altri nodi.

- **Deployment**: lavora ad un livello più alto rispetto ai ReplicaSet, permettendo così di gestire il versioning dell'applicazione attraverso update e rollback.

2.2.2 CRD e Operatori

Uno dei grandi vantaggi di Kubernetes è la possibilità di estendere le funzionalità base creando delle Custom Resource attraverso dei file yaml chiamati Custom Resource Definition (abbreviato in CRD). Una CRD permette di creare un nuovo tipo di oggetto utile per la propria applicazione e di aggiungerla all'API server per poterlo utilizzare.

In caso di applicazioni o configurazioni più complesse, che richiedono la creazione di più risorse, si fa spesso uso degli operatori. Un operatore è un controller che per prima cosa si occupa di effettuare il deploy delle CRD create dall'utente e di introdurne a sua volta altre. Una volta avviata l'esecuzione dell'applicazione, l'operatore ha anche il compito di verificarne lo stato, di ricreare le risorse in caso di failure, di effettuare update, backup e di gestire la scalabilità.

2.2.3 Applicazioni stateful

Di default, essendo Kubernetes basato sui container, tutte le applicazioni perdono i dati alla distruzione del container stesso. Per garantire la persistenza dei dati, si fa uso di due importanti risorse:

- **Persistent Volume (PV)**: è uno storage fornito dall'amministratore del cluster o da una Storage Class e ha un ciclo di vita indipendente dal resto delle risorse esistenti.

- **Persistent Volume Claim (PVC)**: attraverso un PVC un pod richiede di avere accesso a un PV specificando la quantità di storage richiesto e la modalità di accesso, che può essere ReadWriteOnce, ReadOnlyMany o ReadWriteMany.

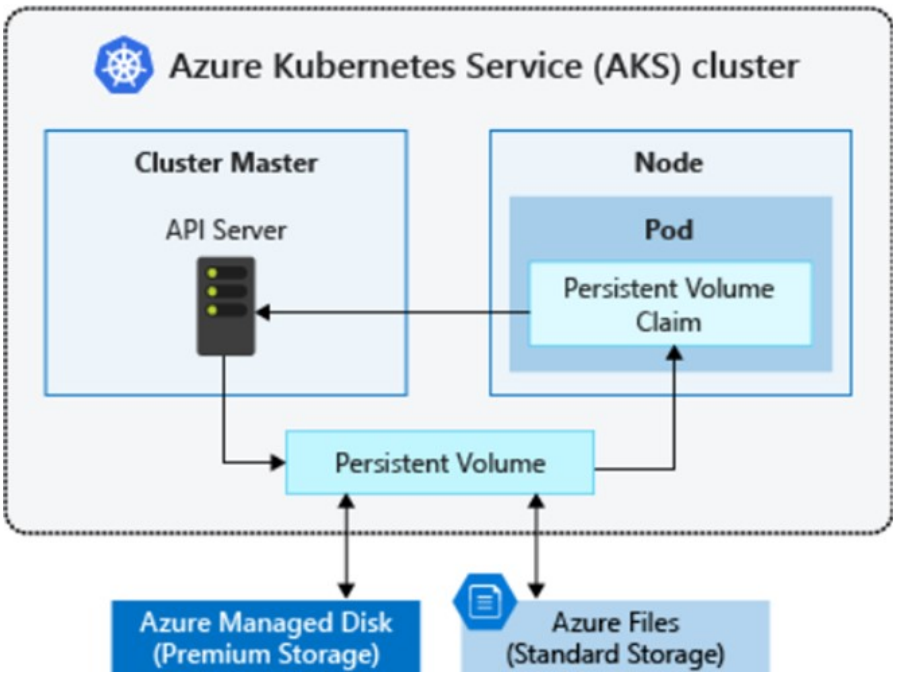

Figura 2.1. Architettura di un'applicazione stateful su Azure [5]

2.3 GitHub

In questa tesi vengono presi in analisi prevalentemente progetti open source, in continua evoluzione e seguiti da community molto attive. Per tale motivo, un ruolo centrale è stato ricoperto dalla piattaforma per lo sviluppo collaborativo di software più celebre: GitHub.

Il principale utilizzo della piattaforma è il controllo di versione, il quale permette di effettuare l'upload di nuovi aggiornamenti nel codice senza sovrascrivere le versioni precedenti, garantendo così la possibilità di eseguire rollback o di utilizzare release precedenti del codice ad esempio per problemi di compatibilità.

Come riportato in precedenza, nella crescita di un progetto open source è fondamentale l'intervento della community, e per questo aspetto una sezione del portale GitHub che permette un'interazione diretta tra la community e gli sviluppatori è la sezione Issues: ogni utente che riscontra un problema o bug nell'esecuzione del codice può aprire una segnalazione tramite questa sezione, potendo così interagire con gli sviluppatori per apportare migliorie al software.

GitHub fornisce anche degli strumenti per poter valutare l'apprezzamento del progetto. I principali sono:

- **commit**: i commit permettono di creare uno snapshot dell'intera repository in un determinato momento. Il numero di commit è indicativo perché all'aumentare di questo numero è possibile associare un avanzamento del progetto, sia in termini di aggiunta di feature sia in termini di bug fixing.

- **watch**: attraverso la funzione di watch in una repository viene inviata una notifica ogniqualvolta venga effettuata una richiesta di pull o venga aperta una nuova issue.
- **stars**: le stars permettono ad un utente di mostrare apprezzamento a una repository, e permettono quindi di valutare la popolarità di un progetto.
- **fork**: il fork corrisponde a una copia del repository, ma con la possibilità di inviare una pull request dalla repository derivata a quella originale, e viceversa di propagare le modifiche dalla repository originale a quella derivata.
- **contributors**: i contributors sono gli utenti che hanno attivamente contribuito al progetto eseguendo commit.

2.4 Azure e AKS

2.4.1 Introduzione ad Azure

Azure è la piattaforma di cloud di Microsoft ed è una delle tre principali soluzioni di cloud pubblico, insieme ad Amazon AWS e Google Cloud.

Lanciato nel febbraio 2010, Azure fornisce un modello pay-per-use, ovvero i costi dipendono esclusivamente dalle risorse e dai servizi utilizzati. Uno dei grandi vantaggi di una soluzione cloud come Azure infatti è non solo la possibilità di creare macchine virtuali e storage, ma anche di sfruttare una serie di servizi che garantiscono alle aziende sicurezza ed efficienza.

2.4.2 AKS e la configurazione del cluster Kubernetes

Il servizio che è stato utilizzato nell'ambito di questo lavoro di tesi è Azure Kubernetes Service (AKS). AKS permette di semplificare il deploy di un cluster Kubernetes occupandosi autonomamente della configurazione dei nodi, della manutenzione e dei controlli di integrità. Per monitorare lo stato di salute del cluster, vengono generati dei log accessibili all'utente.

Il primo requisito per poter creare un cluster su AKS è possedere una sottoscrizione Azure: una sottoscrizione corrisponde a un contenitore delle risorse create ed è necessaria per poter calcolare i costi da addebitare all'utente. Una volta selezionata la sottoscrizione desiderata, è possibile creare un resource group o utilizzarne uno esistente, ovvero un insieme di risorse che condividono lo stesso ciclo di vita e permessi. Si ha poi la possibilità di assegnare un nome al cluster, selezionare una Azure region in cui effettuare il deploy e il numero di availability zones, per garantire che il cluster non abbia problemi anche in caso di failure fisici di uno dei datacenter. Tra gli ulteriori parametri della configurazione base del cluster è possibile selezionare la versione di Kubernetes da utilizzare, il numero di worker nodes, in quanto il master node è gestito autonomamente da AKS ed è gratuito, e soprattutto la node size. Con node size si intende la tipologia di macchina virtuale che deve eseguire il compito di nodo. Tale macchina virtuale è caratterizzata dal numero di vCPU, dalla dimensione della RAM e dalle prestazioni dello storage, con un costo mensile che cresce all'aumentare delle performance della macchina stessa. Infine, è anche

possibile avere un maggiore dettaglio di configurazione su node pools, autenticazione e autorizzazione, networking e integrazione con altri servizi di Azure come ad esempio il monitoring.

Una volta creato il cluster Kubernetes, esistono vari modi per interagire con esso. Il primo metodo prevede l'utilizzo della cloud shell direttamente dal portale Azure. Una seconda soluzione prevede l'utilizzo della Azure CLI (Command Line Interface). Inoltre è possibile installare kubectl sul proprio terminale e utilizzare il file di configurazione del cluster.

Parte II

Storage distribuito

Capitolo 3

Analisi soluzioni di storage distribuito

3.1 Fio

Scopo della tesi è anche quello di confrontare le performance del sistema di storage in diversi scenari. È quindi necessario utilizzare uno strumento affidabile che garantisca la ripetibilità del test.

Fio [4] è un software open source multipiattaforma realizzato da Jens Axboe in linguaggio C per riuscire a simulare operazioni di lettura e scrittura senza dover scrivere per ogni diverso scenario uno specifico codice.

In particolare, il punto di forza di Fio è la possibilità di utilizzare dei file di configurazione: ogni file di configurazione può essere suddiviso in sezioni, e ogni sezione è caratterizzata da una serie di parametri per personalizzare un test. In questo modo, è possibile avviare Fio selezionando uno specifico test dal file di configurazione.

Ad esempio, con il seguente comando:

```
fio -section=seq-read ./fio-example-config.cfg
```

è possibile eseguire un test utilizzando i parametri indicati nella sezione *seq-read* all'interno del file di configurazione *fio-example-config.cfg*.

Maggiori informazioni sui parametri configurati nei test effettuati verranno fornite successivamente nel paragrafo 6.3.

3.2 Soluzioni esistenti

Sul mercato sono presenti numerose soluzioni, molte delle quali open source. Ne vengono analizzate alcune, concentrandosi sull'architettura, sulle possibilità per l'accesso ai dati e sui diversi vantaggi e svantaggi che caratterizzano ogni soluzione.

3.2.1 Ceph

Ceph è un sistema di storage distribuito dotato di un proprio file system (CephFS) che funziona su sistemi Linux e Windows, su quest'ultimo solo utilizzando iSCSI. Ceph è un progetto open source basato su RADOS (Reliable Autonomic Distributed Object Store), il quale permette di utilizzare un unico cluster di storage per tutte le applicazioni, qualsivoglia tipologia di storage (a oggetti, blocchi o file system) esse utilizzino.

Implementazione

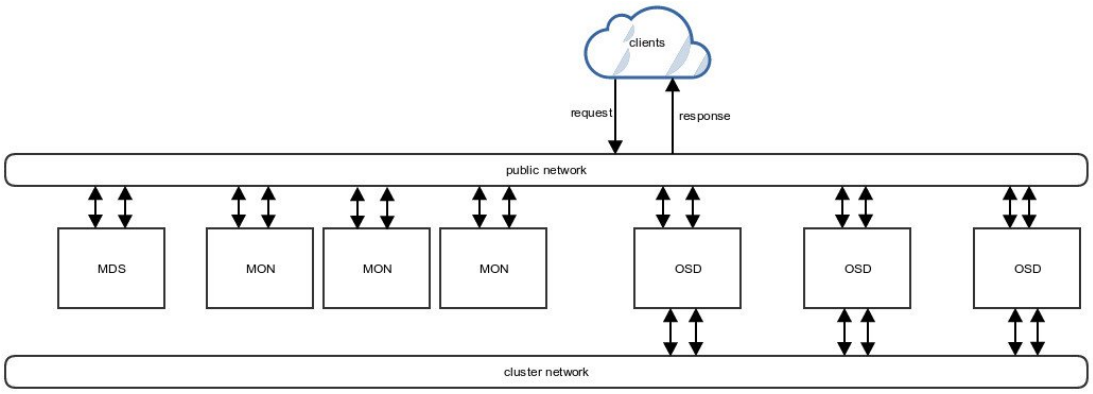

Figura 3.1. Architettura di un cluster Ceph [13]

I nodi che compongono il cluster Ceph possono e devono rivestire determinati ruoli.

- **Manager**: si occupa di gestire lo stato dell'utilizzo della memoria, del carico del sistema e dell'utilizzo dei nodi.
- **Ceph OSD (Object Storage Devices)**: si occupano della memorizzazione, duplicazione e ripristino dei dati, ed è consigliato averne almeno tre nel cluster.
- **Metadata server (MDS)**: si occupano di archiviare i metadati come nome del file, percorso di archiviazione e timestamp. Sono conformi a POSIX. Nel caso di utilizzo di file system forniscono un servizio che mappa i nomi delle cartelle e dei file del file system a degli oggetti salvati nel cluster.
- **Monitor Nodes**: si occupano di gestire lo stato dei singoli nodi del cluster ed è consigliato averne almeno tre nel cluster.

Il funzionamento di Ceph è basato su un algoritmo chiamato CRUSH (Controlled Replication Under Scalable Hashing). Questo algoritmo si occupa sia della distribuzione dei file sia del ritrovamento. La distribuzione dei file avviene calcolando la posizione di archiviazione in base a criteri stabiliti dall'amministratore di rete. La posizione dei file

viene memorizzata in una tabella di assegnazione detta mappa CRUSH, la quale mette in relazione il file e l'OSD che lo contiene. I file sono suddivisi in Placement Groups, dove il nome del file viene elaborato con una funzione di hash. L'algoritmo CRUSH si occupa anche automaticamente di redistribuire intelligentemente i dati in caso di cambiamenti nel cluster, così da garantire sempre ridondanza , senza necessità di intervento dell'amministratore di rete.

Il journaling, ovvero il memorizzare in cache ogni file fino a quando non è memorizzato correttamente in tutti gli OSD previsti, è utilizzato per garantire la sicurezza dei dati.

Ceph viene utilizzato spesso con OpenStack e Cinder per implementare un'infrastruttura iperconvergente, ovvero la computazione e lo storage vengono collocati nello stesso host. Questa infrastruttura permette sia di ottimizzare le risorse, in quanto viene sfruttata nel migliore dei modi sia la potenza di calcolo sia lo storage di ogni nodo, sia di ottimizzare i costi, in quanto non è necessario acquistare dell'hardware specifico al crescere delle necessità. Con OpenStack si può gestire sia un sistema overcloud (sistema che comprende uno o più controller, nodi che forniscono risorse computazionali, nodi che forniscono storage) con solo nodi iperconvergenti, sia un sistema composto da un mix delle tipologie di nodo (iperconvergenti, storage, calcolo).

Accesso ai dati

Esistono diverse alternative per accedere ai dati memorizzati:

- **librados**: è una libreria che permette accesso nativo ai dati tramite delle API scritte in C/C++, Python, Java e PHP. Fornisce funzioni avanzate come snapshot e transazioni atomiche.
- **radosgw**: si sfrutta il protocollo HTTP per leggere o scrivere i dati tramite un gateway.
- **CephFS**: si fa accesso al file system interno, conforme a POSIX, attraverso un modulo kernel per i computer che accedono.
- **RADOS Block Device (RBD)**: si utilizzano moduli kernel o sistemi di virtualizzazione per accedere alla memoria orientata ai blocchi. È realizzato sopra librados, ereditandone le funzionalità, e si occupa di memorizza le immagini dei blocchi come oggetti.

È anche possibile accedere ai dati tramite le API Amazon S3.

Vantaggi

Ceph è una soluzione ormai consolidata nell'ambiente cloud per numerose ragioni. I principali punti di forza derivano dalla natura open source del progetto: esso è completamente gratuito, è dotato di una documentazione molto approfondita ed è semplice ricevere supporto in caso di problemi.

Ulteriori punti di forza di Ceph risiedono nell'assenza di single point of failure, nella possibilità di scalare il cluster sia verticalmente sia orizzontalmente, nella ridondanza integrata e nell'alta disponibilità.

Svantaggi

Ceph richiede diversi componenti per funzionare correttamente, quindi è necessario che il cluster sia composto da un numero sufficiente di nodi. Inoltre, l'installazione è relativamente complessa, e l'utente non ha delle informazioni chiare sull'effettivo luogo di archiviazione dei dati.

3.2.2 GlusterFS

GlusterFS è un sistema di storage distribuito utilizzabile gratuitamente che raggruppa lo spazio di archiviazione dei computer collegati in rete e lo utilizza come unità logica. A differenza di Ceph, questa soluzione supporta solo sistemi Linux, in quanto il modulo FUSE (File System in Userspace) necessario per l'accesso alla memoria non è sufficientemente stabile su Windows.

Implementazione

GlusterFS richiede la presenza di almeno tre nodi per poter funzionare correttamente. I dispositivi integrati comunicano tra di loro attraverso il protocollo TCP/IP, formando un trusted pool. La memoria fornita da essi è composta da bricks che vengono uniti in volumi. A loro volta, i volumi vengono montati e utilizzati come supporti dati. È possibile sfruttare i volumi creando dei RAID, duplicando così i dati garantendo ridondanza o maggiore velocità nell'accesso dei dati.

I volumi possono essere di diverso tipo a seconda dei requisiti:

- **Distributed**: è il tipo di default, prevede che i file siano distribuiti in diversi brick nel volume. Non c'è ridondanza dei dati, rendendo più semplice ed economica la scalabilità, ma non garantendo nessuna protezione dalla perdita di dati.

- **Replicated**: viene garantita ridondanza e quindi protezione dalla perdita dei dati creando delle copie esatte dei dati in tutti i brick. Il numero di repliche può essere deciso dal client al momento della creazione del volume. È consigliabile avere i brick su macchine diverse, così da non avere single point of failure.

- **Distributed Replicated**: i file sono replicati solo su alcuni brick, quindi è richiesto che il numero di brick disponibili sia multiplo del numero di repliche richieste. Le repliche sono generate sui brick adiacenti. Questa tipologia è utilizzata quando è richiesta la disponibilità del dato e contemporaneamente la possibilità di mantenere il sistema scalabile.

- **Dispersed**: sono basati sui codici di cancellazione d'errore. I file vengono codificati e divisi in frammenti. Questi frammenti sono divisi tra i brick aggiungendo della ridondanza. Questa tipologia permette di avere un livello di affidabilità configurabile riducendo al minimo lo spreco di spazio. Il numero di brick ridondanti definiscono quanti brick possono andare persi prima di danneggiare il file.

- **Distributed Dispersed**: come nel distributed replicated, i frammenti del file vengono suddivisi solo in un sottoinsieme dei brick, permettendo di mantenere la proprietà di affidabilità ma rendendo più semplice garantire la scalabilità

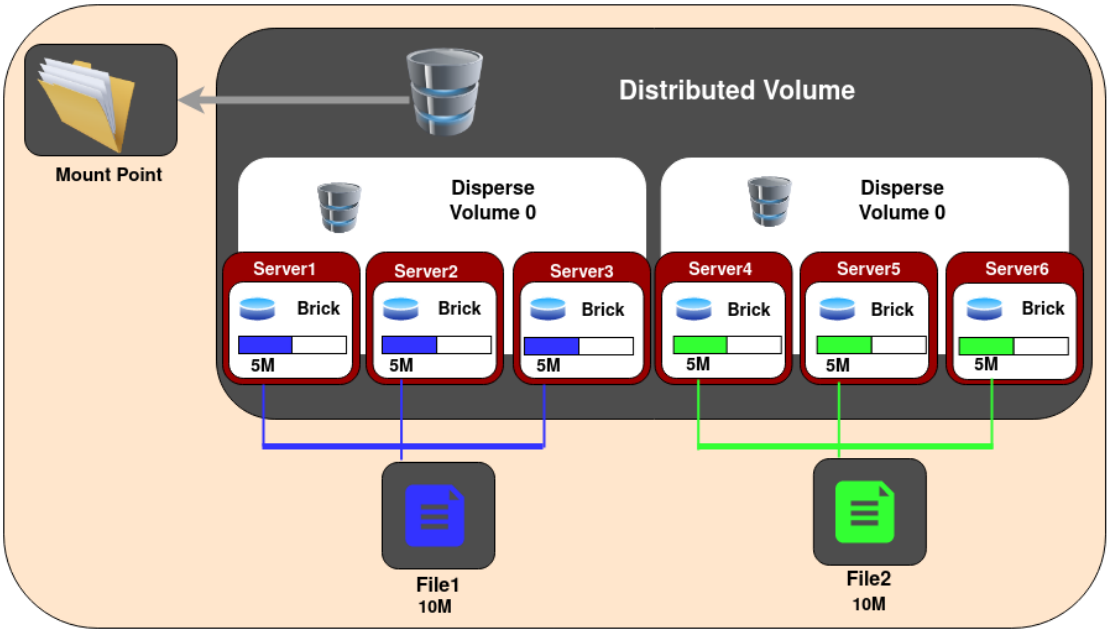

Figura 3.2. Esempio con volume di tipo distributed dispersed [7]

Accesso ai dati

L'interazione con la memoria avviene attraverso i translators, i quali si occupano di convertire le richieste dell'utente in richieste per lo storage, eseguendo se necessario anche cifratura e access control.

I translators sono implementati come oggetti condivisi e ne esistono di 10 tipi: storage, debug, cluster, encryption, protocol, performance, bindings, system, scheduler, features.

In GlusterFS è importante la distinzione tra client e server, e la si incontra sia nel funzionamento dei translators per eseguire una funzione, sia per altre funzionalità come la replica geografica, che permette di duplicare i dati in posizioni geografiche diverse sfruttando la trasmissione di dati tramite SSH.

Vantaggi

Il file system è conforme allo standard POSIX, quindi GlusterFS può esere facilmente integrato in ambienti server Linux eistenti. Utilizzando alberi di file system gerarchici in blocchi, è altamente efficiente per la memorizzazione di file anche di grandi dimensioni.

Grazie alla struttura composta da nodi e bricks, il sistema è facilmente scalabile, essendo solamente necessaria l'aggiunta di unità di archiviazione.

La sicurezza è garantita tramite ridondanza, sfruttando la tecnologia RAID, utilizzabile anche per fornire maggiore velocità di accesso ai dati a seconda della configurazione scelta.

È gratuito, open source e permette l'utilizzo di hardware commodity.

Svantaggi

Il non essere direttamente compatibile con Windows a causa dell'utilizzo di FUSE è il principale svantaggio di GlusterFS.

Poiché i dati vengono trasferiti tra le unità di archiviazione tramite protocolli di rete, è necessaria un'infrastruttura di rete veloce, e ciò richiede anche degli sforzi maggiori nella creazione della rete stessa e nel garantire sicurezza tecnica.

Essendo un sistema basato su alberi di file, l'object storage non è ancora ottimizzato, quindi è consigliabile andare su soluzioni alternative come Ceph per questo tipo di utilizzo.

3.2.3 LizardFS

LizardFS è un sistema di storage distribuito che combina lo spazio di archiviazione di diversi server Linux in un unico namespace visibile tramite client gratuito su sistemi Unix-like e attraverso un client a pagamento su sistemi Windows.

Implementazione

Poiché creare copie dei file per garantire ridondanza non è efficiente, i dati vengono divisi in chunk, i quali, insieme a delle parti addizionali chiamate stripes che garantiscono il ripristino di parti mancanti, vengono distribuiti sui server, migliorando affidabilità ed efficienza. I server si dividono in due tipologie:

- **master server**: contiene i metadati e l'effettiva locazione dei chunk che compongono un file. Viene utilizzato per ridurre la latenza. Si occupa di controllare costantemente se i chunk sono ancora tutti disponibili, così da mantenere i metadati aggiornati anche in caso di malfunzionamento di uno o più chunk server. La ridondanza del master server è garantita attraverso la presenza di uno shadow master server, che rimane sincronizzato con il master server attivo e lo sostituisce in caso di problemi.

- **chunk server**: sono i server che contengono effettivamente i chunk di dati. Possono sfruttare hardware commodity. Lo storage può aumentare o diminuire semplicemente aggiungendo o rimuovendo server, in quanto LizardFS si occuperà in automatico di spostare i chunk tra i dischi presenti, garantendo ridondanza dei dati e bilanciamento nell'occupazione dello spazio. Problemi nei dischi e nei server sono gestiti senza downtime e perdita di dati.

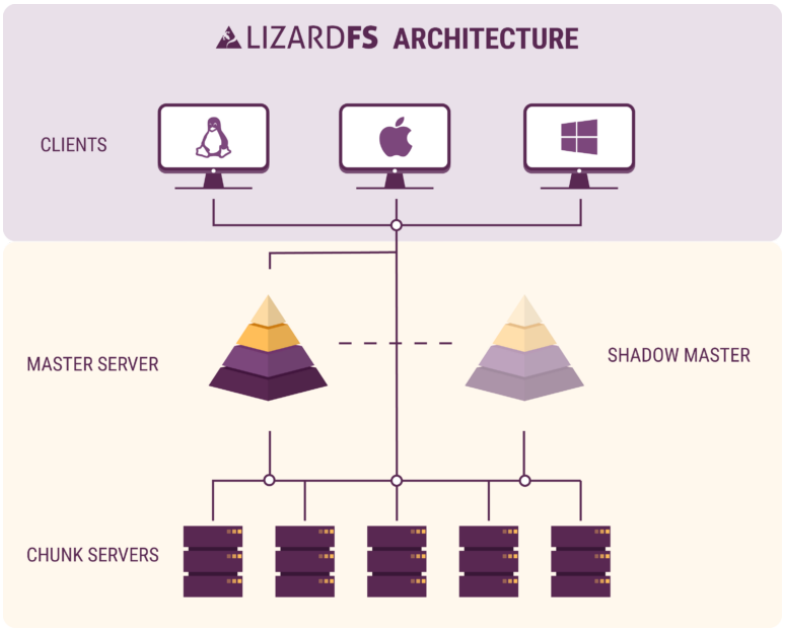

Figura 3.3. Architettura di un cluster LizardFS [8]

Quando un client deve fare accesso a un dato, contatta prima il master server per ottenere i metadati del file richiesto, in seguito contatta in parallelo tutti i chunk server che contengono le parti del file richiesto e le ottiene in contemporanea, aumentando le performance.

LizardFS sfrutta il principio del CoW (copy-on-write) attraverso gli snaphot, ovvero in caso di richiesta di copia di un file, vengono copiati solo i metadati, così da velocizzare l'operazione. I dati verranno duplicati solo successivamente in caso di modifica.

È fornito un cestino che in caso di necessità permette all'amministratore di recuperare un file eliminato erroneamente da un utente.

Accesso ai dati

LizardFS è conforme allo standard POSIX. Vengono forniti client nativi per Linux, OSX e Windows. Una volta montato, LizardFS comparirà come mount point sui sistemi Linux e OSX, mentre come disco con corrispondente lettera sui sistemi Windows.

Vantaggi

LizardFS è una soluzione molto affidabile ed efficiente, la quale pone molta attenzione sull'evitare un uso eccessivo dello spazio di archiviazione e garantisce ottime performance nell'accesso ai dati di grandi dimensioni. Permette l'utilizzo di hardware commodity e sfrutta gli snapshot per la copia dei file. Tra gli altri vantaggi, LizardFS rende possibile sfruttare la georeplicazione, impostare dei meccanismi di QoS, fissare dei limiti per gli utenti e usare degli strumenti di monitoring.

Svantaggi

Il principale svantaggio consiste nel non essere del tutto gratuito, in quanto il client Windows viene fornito solo se si acquista uno dei tre pacchetti di supporto messi a disposizione.

A causa del suo sistema di sincronizzazione, non è adatto per le modifiche in tempo reale sui file, in quanto porterebbero un continuo trasferimento di dati con conseguente rallentamento del sistema. È quindi suggerito lavorare su un file in una cartella temporanea e, una volta terminata la modifica, copiare il file finale nello storage distribuito.

Le performance nella scrittura di un gran numero di file di piccole dimensioni non sono così elevate.

3.2.4 vSAN

vSAN è una soluzione di storage software-defined implementata da VMWare. Essa permette di astrarre e aggregare dei dischi in un cluster vSphere che siano gestibili attraverso vCenter e accessibili ai client vSphere.

Implementazione

Ogni host può avere una configurazione "all flash", ovvero utilizzare solo SSD come dispositivi di archiviazione, o una configurazione "ibrida" composta da SSD e HDD. I dispositivi di archiviazione sono raggruppati in "disk groups", i quali possono essere in un numero tra uno e cinque per ogni host. Ogni disk group è composto da un SSD che fa da cache e da uno a sette dispositivi di archiviazione che contengono i veri e propri dati.

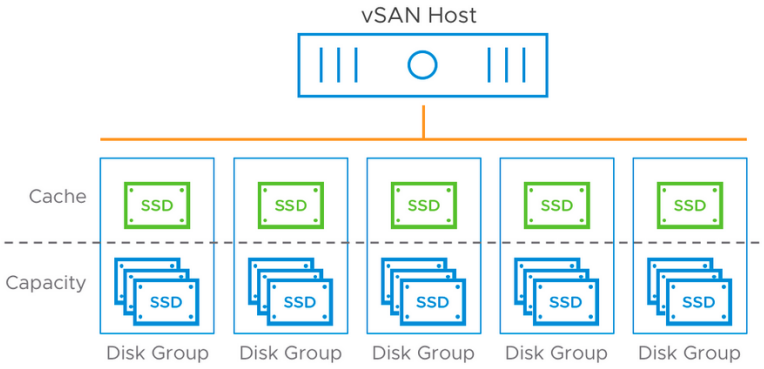

Figura 3.4. Architettura di un data store in un cluster vSAN [16]

I cluster vSAN possono essere di tre tipi:

- **Standard cluster**: è formato da tre a 64 nodi fisici situati nello stesso luogo e collegati da una rete a 10 Gbps o superiore.

- **2 node cluster**: è composto da coppie di nodi direttamente connessi o connessi tramite lo stesso network switch. Ogni coppia di nodi richiede un host "witness" che permetta di raggiungere una decisione in caso di informazioni diverse provenienti dai due nodi. Lo stesso host witness può essere condiviso da un massimo di 64 coppie di nodi.
- **Stretched cluster**: gli host sono distribuiti in due luoghi diversi, garantendo ridondanza nel caso avvenga un problema che coinvolga un intero sito. I due siti devono essere collegati garantendo una latenza inferiore ai 5ms. Anche in questa struttura è presente un host vSAN in un terzo sito. Si possono avere fino ad un massimo di 30 host per cluster.

La peculiarità di vSAN è quella di essere perfettamente integrabile in un'architettura iperconvergente. Essa infatti viene utilizzata con VMware vSphere per sfruttare dei nodi che comprendono sia capacità di calcolo che capacità di storage. Per aumentare l'agilità del sistema, VMware ha rilasciato vSAN HCI Mesh, ovvero un approccio che permette di raggruppare più cluster vSAN. Ciò permette di condividere dati, di trasferire VM e, in generale, di scambiare informazioni tra i cluster in maniera molto più semplice ed efficiente, senza richiedere strumenti e hardware esterni. HCI Mesh permette anche a cluster non dotati di storage di accedere ai dati condivisi da altri cluster, e questo è un grande vantaggio in quanto è possibile aumentare la capacità computazionale senza sostituire server già esistenti.

Accesso ai dati

A seconda della configurazione scelta per i dischi, l'accesso in scrittura e lettura avviene a velocità diversa. In particolare, la scrittura viene sempre fatta nel disco di cache, quindi la VM che deve scrivere eseguirà l'operazione sempre alla velocità del dispositivo di cache. Per quanto riguarda la lettura invece, nel caso in cui il dato non sia presente in cache, si hanno dei tempi superiori nel caso in cui si stia usando una configurazione ibrida, a causa della minore velocità dei dischi magnetici rispetto agli SSD.

Vantaggi

Usare una soluzione proprietaria ha il grande vantaggio di avere garantito un supporto completo, riducendo il tempo necessario nella risoluzione dei problemi e nella configurazione iniziale.

vSAN è incluso nell'hypervisor vSphere, di conseguenza l'utilizzo delle risorse è ottimizzato, e non sono richieste delle macchine virtuali che devono occuparsi esplicitamente dello storage. Ciò comporta un percorso di I/O più corto, un minor utilizzo di risorse e di conseguenza un risparmio in termini economici. Insieme alla suite VMware, fornisce dei comodi strumenti di monitoring.

Svantaggi

vSAN non è una soluzione openSource, di conseguenza bisogna tener conto dei costi delle licenze. Inoltre è una soluzione adatta solo se si utilizza la suite di prodotti VMware.

I requisiti hardware possono essere un vincolo notevole, soprattutto nel caso si sia provvisti solo di hardware più antiquato per quanto riguarda i dispositivi di rete. Le performance di disaster recovery non sono molto elevate e l'aggiornabilità è spesso complessa.

3.2.5 MinIO

MinIO è un sistema di storage distribuito open source basato sugli oggetti. È un progetto cloud-native orientato a Kubernetes e offre una licenza commerciale a costo mensile per ricevere supporto e aiuto negli aggiornamenti. Il suo punto di forza riguarda la memorizzazione di file come immagini, video, audio.

Implementazione

Un container di MinIO pesa circa 40 MB ed è molto efficiente nell'uso della CPU e della memoria. Può essere eseguito su hardware commodity al quale sono collegati direttamente dei dispositivi di archiviazione. Tutti i nodi del cluster hanno lo stesso funzionamento, senza nessuna distinzione di server che si occupano di metadati, in quanto i metadati vengono memorizzati insieme ai dati stessi. MinIO gira come processo in user space e si occupa di eseguire operazioni di cifratura, controllo di integrità e ridondanza dei dati con delle operazioni inline scritte in codice assembly, così da renderlo il più efficiente possibile. Il livello di ridondanza può essere personalizzato dall'utente. I server MinIO possono essere raggruppati in un Distributed Mode set, il quale può essere raggruppato con altri Distributed Mode set in una MinIO Server Federation, la quale è gestita da un unico admin e ha un unico namespace. Questa organizzazione rende più semplice la scalabilità del sistema.

Accesso ai dati

I singoli nodi comunicano tra di loro (per scambiare dati al fine di garantire ridondanza ecc.) attraverso delle API RESTful. Le applicazioni invece utilizzano le API di Amazon S3 e ciò permette di interagire con infrastrutture dati a file o a blocchi come se fossero a oggetti, unificando l'interazione. L'utilizzo delle API S3 permettono anche di rendere MinIO un'architettura multi-cloud, in quanto è possibile eseguirla su un cloud privato, su un cloud pubblico o su dell'hardware on-premise.

Vantaggi

MinIO è una delle soluzioni di storage distribuito più diffuse, grazie al suo essere open source, alla sicurezza nella gestione dei dati grazie agli algoritmi di cifratura utilizzati, alla comodità nell'utilizzo delle API S3 per l'interazione applicazioni-storage. Inoltre le performance sono molto elevate, e la possibilità nel mondo enterprise di poter gestire una struttura multi-cloud è un notevole pro.

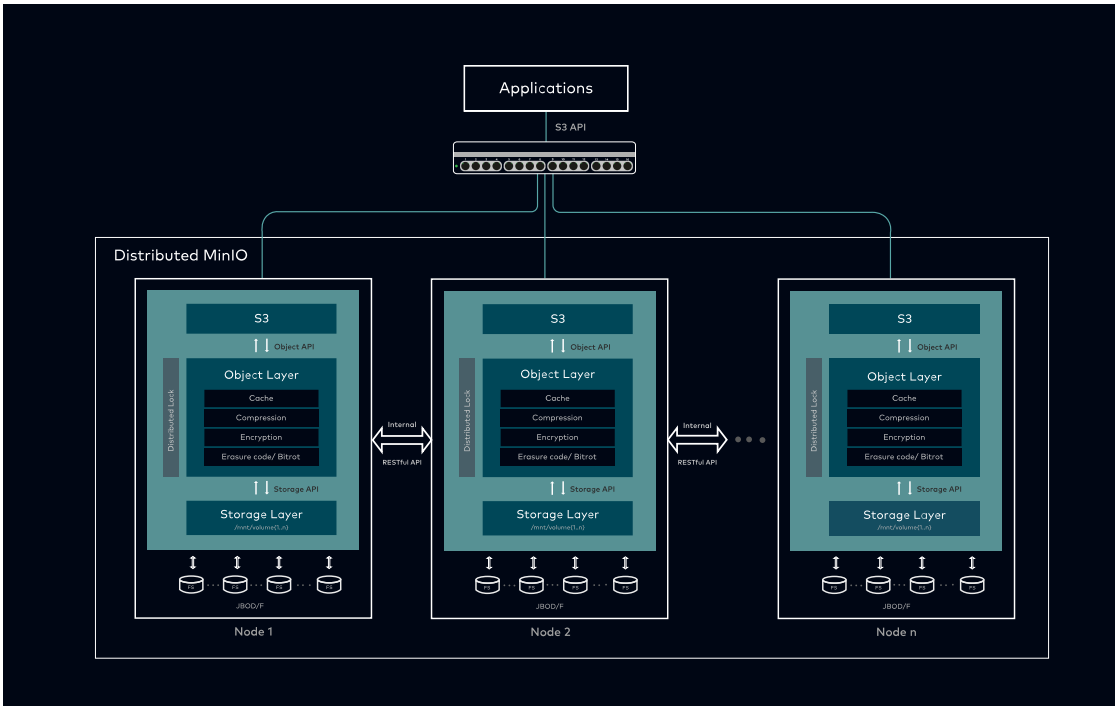

Figura 3.5. Architettura di un cluster MinIO [12]

Svantaggi

Non offre dei comodi strumenti di monitoring. Nasce come storage orientato agli oggetti, quindi non si ha supporto nativo per i file system e i sistemi a blocchi.

3.2.6 OpenIO

OpenIO è una soluzione di storage distribuito open source che punta sulle performance e sulla scalabilità. Così come MinIO, anche in questo caso è possibile acquistare dei pacchetti per avere accesso al supporto e a dei tool specifici.

Implementazione

OpenIO può essere eseguito su qualsiasi hardware ed è un sistema orientato agli oggetti. Uno dei principali punti di forza di OpenIO è l'essere progettato come griglia di nodi invece di avere una struttura ad anello. Questa conformazione permette di rendere il cluster più flessibile e di avere una distribuzione più efficiente e sicura dei dati tra i nodi. Inoltre, a differenza di altri sistemi di storage distribuito, quando viene aggiunto un nodo non è necessario effettuare la redistribuzione dei dati, operazione onerosa in termini di risorse e di tempo. OpenIO infatti, una volta aggiunto un nuovo nodo, si occuperà

solamente di analizzarne la capacità e le prestazioni, e lo utilizzerà quando necessario per sfruttarlo al massimo. La tecnologia che permette di far ciò è denominata ConsciousGrid. Questa organizzazione rende molto più efficiente la scalabilità del sistema, permettendo di sfruttare sempre il più possibile lo spazio a disposizione e garantendo delle buone prestazioni anche quando si esegue manutenzione sul cluster aggiungendo o rimuovendo nodi.

Viene garantita la sicurezza dei dati tramite cifratura, viene garantita la geo-replica anche per connessioni con alta latenza e poca banda, e viene sfruttato l'erasure coding per ottimizzare la gestione dello spazio senza sacrificare la ridondanza. Grazie all'utilizzo di strumenti basati su Prometheus e Grafana, si hanno a disposizione anche dei tool per il monitoring. Una feature che altre soluzioni non hanno è chiamata GridForApps, e permette di catturare gli eventi all'interno del cluster e passarli a delle applicazioni all'interno dei nodi, così da poter effettuare elaborazione, indicizzazione sui metadati, applicativi di machine learning ecc.

Accesso ai dati

Per interagire con OpenIO si può fare uso delle API Amazon S3, ma viene anche fornita una CLI, supporto ad Ansible e, a pagamento, una WebUI. Per le applicazioni che hanno bisogno di accesso avanzato a funzioni di storage, sono previste delle API native REST/HTTP e delle API per Python, C/C++ e Java, oltre alla compatibilità con le API di Openstack Swift.

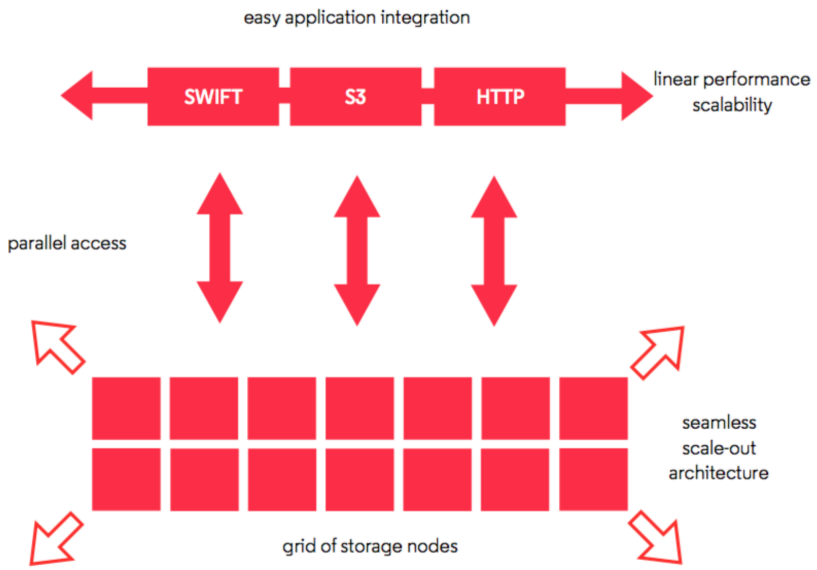

Figura 3.6. Accesso ai dati in un cluster OpenIO [14]

Vantaggi

OpenIO è una soluzione che punta molto sulle performance nel trasferimento dei dati e nella reattività del sistema in seguito a operazioni di scalabilità e manutenzione. Sono dei punti a favore anche la sicurezza nel mantenimento dei dati, le numerose API utilizzabili per accedere ai dati stessi e la possibilità di utilizzare qualsiasi hardware, oltre alla natura opern source del progetto.

Svantaggi

Alcune funzionalità aggiuntive come la WebUI e gli strumenti di sviluppo sono presenti solo nel pacchetto a pagamento.

3.2.7 HDFS

HDFS (Hadoop Distributed File System) è un sistema di storage distribuito open source appartenente al pacchetto Hadoop. È progettato per essere eseguito su hardware economico garantendo comunque affidabilità e ridondanza.

Implementazione

HDFS è progettato per il batch processing piuttosto che per un utilizzo interattivo da parte degli utenti. Per questo motivo, la sua implementazione punta a favorire un elevato throughput nell'accesso dei dati a discapito della latenza. Per la stessa ragione, si ottengono delle ottime performance con file di grandi dimensioni e prestazioni meno buone per file piccoli.

L'architettura è di tipo master-slave, quindi i nodi posso avere due diversi ruoli:

- **NameNode**: è il master server che si occupa di memorizzare i metadati dei file e la posizione dei chunks all'interno dei nodi. Il suo ruolo è anche quello di ricevere dai nodi le informazioni sul loro stato e sullo stato dei dati, così da poter avviare le procedure di duplicazione dei dati per garantire sempre ridondanza.
- **DataNode**: sono i nodi direttamente collegati alle unità di archiviazione, si occupano di eseguire la creazione e la duplicazione dei chunks seguendo le istruzioni del NameNode, e servono le richieste di lettura e scrittura dai client.

Qualsiasi macchina che supporti Java può essere utilizzata come nodo. È raro utilizzare la stessa macchina per eseguire più di un nodo.

HDFS si basa su un'organizzazione tradizionale a file system gerarchico, è quindi possibile creare, rimuovere, rinominare e spostare tra le cartelle i file. Tutte queste informazioni sono contenute in dei file gestiti dal NameNode.

HDFS garantisce ridondanza e il numero di repliche dei file può essere personalizzato dall'utente sia alla creazione del file stesso sia successivamente.

Per favorire le performance viene sfruttata la funzionalità della replica geografica sia all'interno del cluster sia tra i diversi cluster, scegliendo sempre la replica del dato più vicina all'applicazione che ne deve fare uso.

È possibile fare uso di snapshot per ripristinare lo stato di un dato nel caso in cui si danneggi.

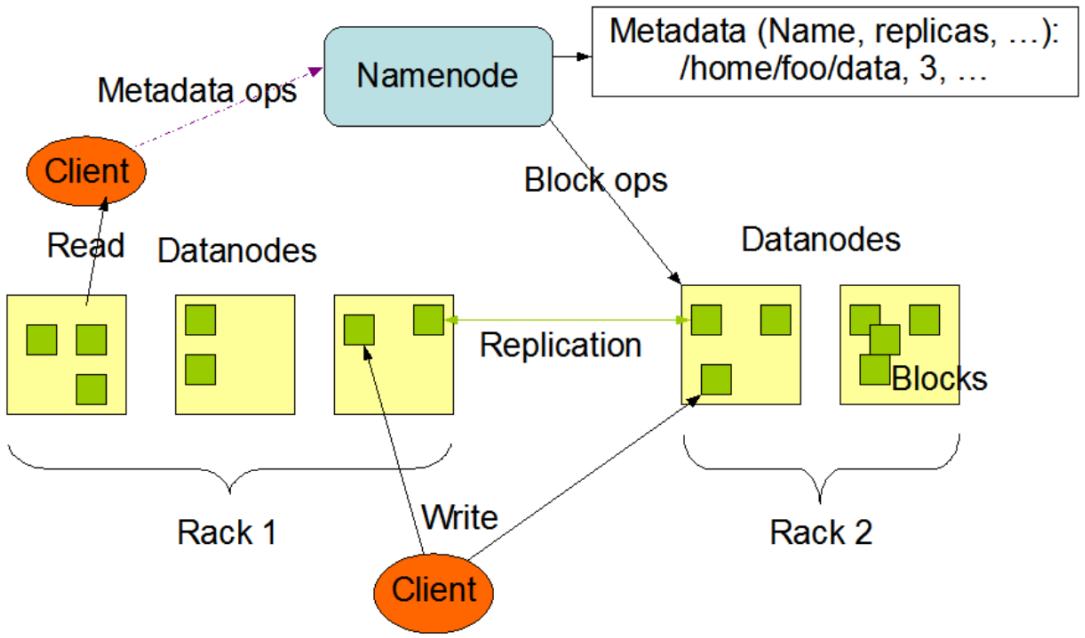

Figura 3.7. Architettura di un cluster HDFS [1]

Accesso ai dati

La comunicazione tra il client, il NameNode e i DataNodes avviene attraverso il protocollo TCP/IP. Le applicazioni hanno diversi modi per accedere ai dati: il metodo nativo consiste in delle API Java, ma sono presenti anche delle API wrapper in linguaggio C e delle API REST. È anche possibile accedere allo storage sia attraverso un browser HTTP, con la possibilità di sfogliare i file e interagirvi, o anche visualizzando i file dal file system locale del client attraverso un gateway NFS.

Infine, viene fornita una CLI chiamata "FS shell" per garantire all'utente l'interazione diretta con il file system o per le applicazioni che hanno necessità di un linguaggio di scripting per interagire con i dati memorizzati.

Vantaggi

I principali punti di forza di HDFS consistono nella possibilità di utilizzare hardware commodity non costoso, nella facilità di utilizzo, nelle performance per i file di grandi dimensioni. È anche un sistema facilmente scalabile, in quanto la struttura master-slave permette di variare la capacità di archiviazione semplicemente aggiungendo o rimuovendo

DataNodes. La creazione di repliche dei dati distribuite su nodi diversi, con la possibilità di avere i nodi distribuiti su più cluster, lo rende anche molto affidabile.

Svantaggi

HDFS ha diversi svantaggi intrinseci, dovuti alla sua architettura. Uno di questi è dato dalle performance con file di piccole dimensioni: essendo basato sul batch processing, ovvero una grande quantità di dati viene letta, elaborata e viene poi prodotto un risultato, non è adatto per applicazioni che richiedono un'elaborazione dei dati in tempo reale. La latenza è un altro punto a sfavore di HDFS riconducibile al batch processing.

Il principale problema però si trova nella sicurezza, in quanto HDFS non prevede cifratura dei dati nello storage e nella trasmissione dei dati, e inoltre si basa su Java, che essendo molto diffuso è anche facilmente attaccabile.

3.2.8 Lustre

Lustre è un sistema di storage distribuito open source basato sulla parallelizzazione. Quest'ultimo aspetto gli permette di raggiungere delle performance tali da essere il sistema di storage più utilizzato nell'HPC (High Performance Computing). Questi computer infatti hanno bisogno di elaborare enormi quantità di dati e file di grande dimensione.

Implementazione

Lustre viene eseguito su sistemi Linux in una struttura client-server.

I server possono ricoprire dei ruoli diversi:

- **Object Storage Target (OST)**: è costituito da un dispositivo di archiviazione o da più di uno in RAID. Ogni file è memorizzato in uno o più oggetti, con ogni oggetto collocato in un OST. Il numero di oggetti in cui deve essere diviso un file è configurabile dall'utente per raggiungere le migliori prestazioni possibili.

- **Object Storage Server (OSS)**: si occupa di gestire gli accessi e le richieste di rete di un gruppo di OST (tra 2 e 8). Contiene alcuni metadati sui file contenuti negli OST che gestisce.

- **Metadata Server (MDS)**: è un nodo che tiene traccia della posizione di ogni file così da poter indirizzare le richieste del client al corrispondente OSS e di conseguenza agli OST che contengono effettivamente il dato richiesto. Dopo aver individuato la posizione, l'MDS non interagisce più con l'I/O del file.

- **Metadata Target (MDT)**: contiene tutti i metadati necessari all'MDS per eseguire le proprie operazioni. Questa divisione tra MDT e MDS consente una divisione delle risorse necessarie per l'elaborazione da quelle necessarie per la memorizzazione.

- **Management Service (MGS)**: è un nodo che si occupa di verificare lo stato degli altri nodi e di gestire la configurazione del sistema.

- **Management Service Storage Target (MGT)**: memorizza tutti i dati di configurazione.

Lustre ha un'architettura di tipo gerarchico, si basa infatti su un insieme di tecnologie e processi chiamato Hierarchical Storage Management (HSM) che prevede la presenza di un archivio che contiene una copia dei file usati più frequentemente, così da garantire un accesso più rapido da parte delle applicazioni. Quando termina lo spazio disponibile nell'archivio, la copia del file viene rimossa, per venire recuperata dall'OSS corrispondente quando arriverà una nuova richiesta di accesso da parte del client. Questa architettura, oltre a garantire migliori performance, permette lato client di utilizzare le stesse system calls per accedere al dato, qualunque sia la sua posizione.

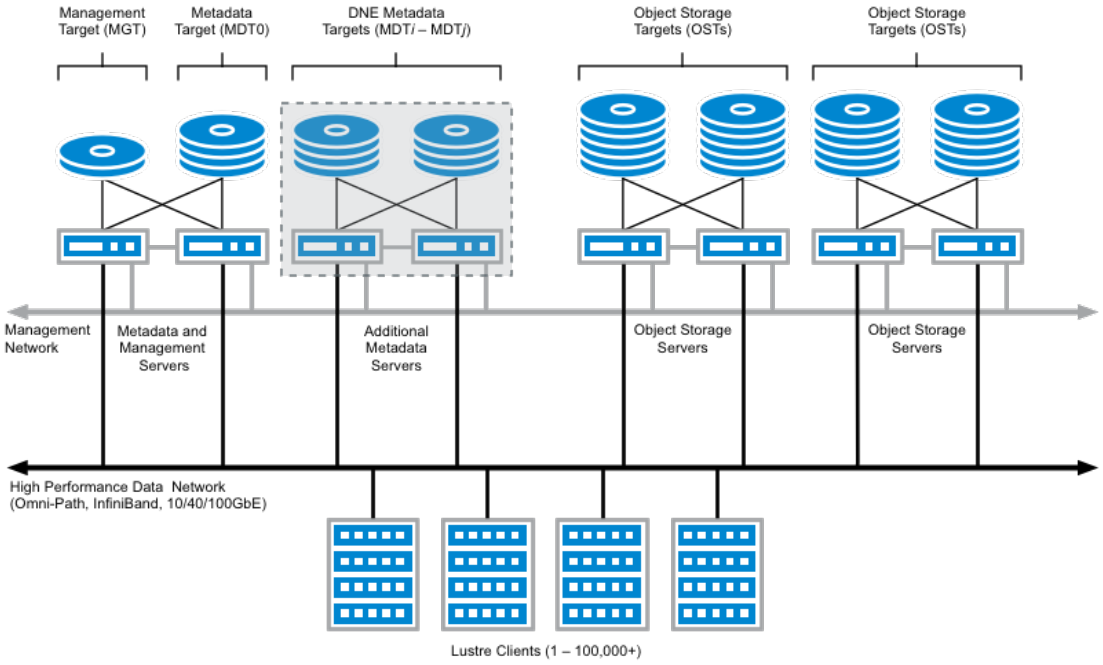

Figura 3.8. Architettura di un cluster Lustre [10]

Accesso ai dati

Il sistema è conforme allo standard POSIX. Il client sfrutta una connessione di rete per comunicare utilizzando delle RPC (remote procedure call). Vengono stabilite delle connessioni con il MGS per la configurazione, con il MDS per richiedere i metadati del file richiesto e con l'OST per recuperare il file.

Sono disponibili dei componenti aggiuntivi (e.g. Integrated Manager for Lustre) che introducono delle API REST e delle CLI per fornire ulteriori metodi di interazione con il file system.

Vantaggi

Lustre è la soluzione ideale per fornire uno storage distribuito in grado di essere efficiente, scalabile e accessibile anche in presenza di numeri molto grandi di dati o se bisogna lavorare con file molto grandi, come nel caso dei computer più veloci. L'architettura permette di garantire ridondanza dei nodi, di eseguire manutenzione e di risolvere guasti senza che le applicazioni possano venirne influenzate. Inoltre è un progetto open source e viene fornita assistenza a pagamento.

Svantaggi

Il principale svantaggio è quello che caratterizza tutti i sistemi che si basano sulla divisione di dati e metadati: nel caso in cui un'applicazione, ad esempio nell'ambito dell'IoT, generi tantissimi file di dimensioni molto ridotte, si porta rapidamente a saturazione la memoria per i metadati, causando anche un rallentamento evidente nelle operazioni di elaborazione. È lo stesso problema che si riscontra in applicazioni di AI/ML, che richiedono accesso a grandi quantità di dati di piccole dimensioni con bassa latenza.

3.2.9 SeaweedFS

SeaweedFS è un sistema di storage distribuito open source che si prefigge l'obiettivo di essere veloce e di gestire grandi quantità di dati nella maniera più semplice possibile.

Implementazione

L'idea alla base dell'architettura è quella di tenere dati e metadati in un unico luogo, così da effettuare una sola operazione sul disco per accedere al dato. Per fare questo, SeaweedFS si basa sulla gestione dei volumi, e ogni nodo può contenerne più di uno.

In particolare, i nodi possono essere di due tipi:

- **Volume Server**: sono i server che contengono effettivamente i dati, mantengono in RAM per ogni file una entry di 16 byte che indica in quale volume è memorizzato il file e con quale offset. È direttamente collegato ai dispositivi di archiviazione che contengono i volumi, i quali possono avere una dimensione massima di 32 GiB.
- **VMaster Server**: si occupa di memorizzare in quale volume server è contenuto un determinato volume, memorizzato attraverso un id definito tramite funzione di hash. Questo ruolo può essere ricoperto da qualsiasi soluzione già esistente come Mongodb, MySql, Postgres ecc. Per evitare che sia un SPOF, è possibile in fase di configurazione crearne più di uno.

Questa architettura permette di sfruttare contemporaneamente i server on-premise con il cloud storage: è possibile infatti utilizzare i volumi locali, più veloci ma dalle capacità ridotte, per memorizzare i dati a cui si fa accesso più di frequente, memorizzando invece sul cloud tutti i file a cui si accede meno.

In fase di configurazione, l'amministratore può decidere di dividere i file di grandi dimensioni in chunks, così da garantire rapidità di accesso anche con file molto grandi. La

ridondanza non è implementata di default, è però possibile utilizzare per ogni nodo degli script che si occupino di implementarla. In particolare, per i dati a cui si fa accesso più di frequente, SeaweedFS prevede la creazione di repliche, mentre per i dati meno utilizzati si utilizza l'algoritmo di erasure coding.

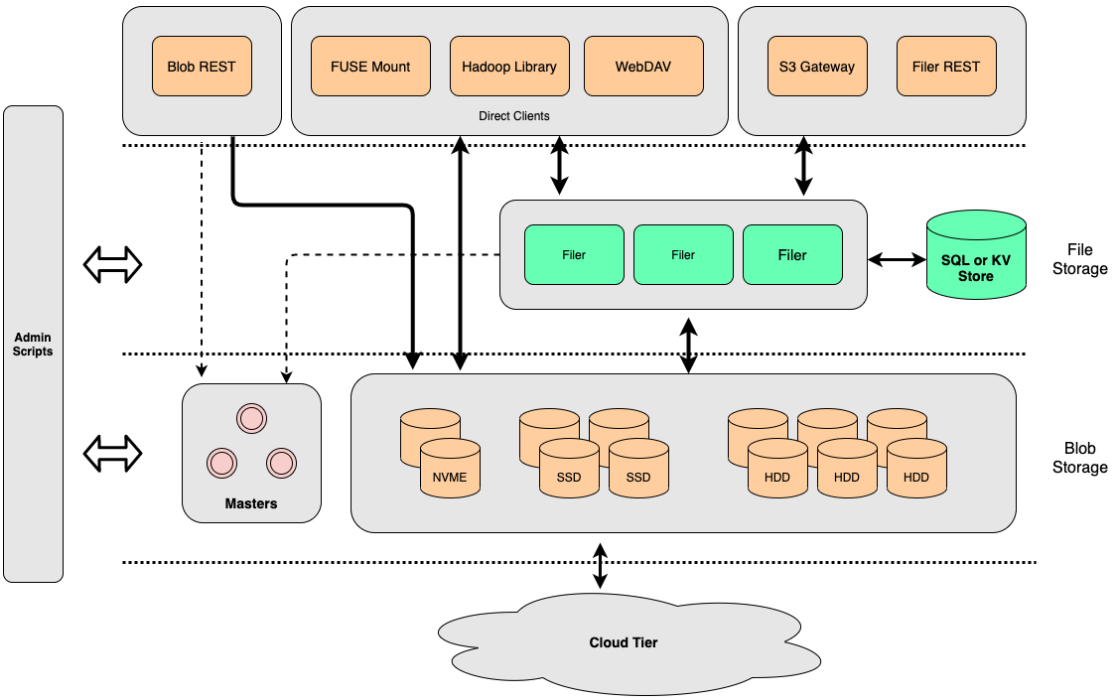

Figura 3.9. Architettura di un cluster SeaweedFS [9]

Accesso ai dati

Le operazioni di lettura e scrittura tra client e server avvengono tramite delle POST e delle GET, scambiando prima con il master server le informazioni sul volume che contiene il dato o che deve memorizzarlo.

L'accesso ai file può essere fatto attraverso le API Amazon S3 e le API REST. Inoltre, tramite SeaweedFS Filer, il sistema diventa conforme allo standard POSIX.

Vantaggi

SeaweedFS è un sistema che promette le principali funzioni di tutti gli altri sistemi di storage distribuito ma attraverso un'implementazione molto più semplice. È ottimizzato per dare ottime performance anche con un gran numero di file di piccole dimensioni, fornendo la possibilità di configurare la divisione in chunks per garantire performance anche con file di grandi dimensioni. Non ha SPOF e garantisce scalabilità in maniera

semplice, senza aver bisogno della ridistribuzione dei dati quando viene aggiunto un nuovo nodo. Garantisce anche la sicurezza dei dati in quanto sono previsti degli algoritmi di cifratura.

Svantaggi

È una soluzione in continua evoluzione, quindi necessita aggiornamenti frequenti.

3.2.10 FreeNAS

FreeNAS è un sistema di storage basato su file system ZFS che ha lo scopo di semplificare ogni task ad ogni livello. Non è un sistema di storage distribuito, ma viene analizzato in quanto attualmente utilizzato dall'azienda. Fornisce supporto per storage di file, a blocchi e a oggetti per tutti i principali sistemi operativi.

Implementazione

FreeNAS utilizza ZFS come file storage. ZFS funge anche da gestore di volumi, occupandosi del partizionamento e della formattazione dei dischi che possono essere aggiunti in ogni momento al pool. Questo file system ha una propria implementazione di RAID chiamata RAID-Z, la quale è una variante di RAID-5 e richiede l'inserimento di unità in multipli di 2.

FreeNAS sfrutta anche l'archiettura copy-on-write di ZFS per garantire che non vi sia perdita dei dati in caso di problemi durante la sovrascrittura di un dato. Il copy-on-write viene utilizzato per la creazione di snapshot dell'intero pool o di singoli blocchi, e che possono essere eseguiti manualmente o in automatico periodicamente. Gli snapshot permettono di ripristinare una versione precedente di un file o del sistema. Nel caso di eliminazione di un file, vengono anche eliminati tutti gli snapshot a esso collegati. Ogni snapshot può essere clonato per creare delle versioni "writable" da usare ad esempio per il deploy di virtual machine basate su una master image. FreeNAS supporta le repliche: uno snapshot può essere copiato in un file system remoto, e viene tenuto sincronizzato inviando solo le modifiche effettuate, rendendo efficiente lo scambio di dati.

L'integrità dei dati è garantita attraverso il calcolo di un checksum al momento della scrittura, il quale viene verificato in fase di lettura. Vengono utilizzati dei bit di parità per recuperare i dati corrotti, che possono essere individuati anche attraverso un controllo mensile che viene effettuato su tutti i blocchi del sistema.

FreeNAS è un sistema che si occupa della ridondanza dei dati fornendo cifratura dei volumi e supportando SED (Self-Encrypting Drives).

Accesso ai dati

Per semplificare la configurazione e l'esperienza utente, FreeNAS fornisce una Web Interface che permette di eseguire tutte le operazioni, dalla configurazione dello storage agli aggiornamenti, insieme a garantire accesso via SSH per la diagnosi e i task amministrativi. Si può fare accesso ai dati attraverso vari protocolli, come ad esempio FTP, Windows SMB, Unix NFS ecc.

Vantaggi

FreeNAS è un sistema molto personalizzabile, sicuro e affidabile. Permette di utilizzare gli SSD come cache per velocizzare sia lettura sia scrittura. Ha un'ottima UI che permette in maniera semplice di gestire i dischi.

Svantaggi

Secondo l'opinione di molti utenti, il primo approccio con molte opzioni, in particolare sulla gestione della sicurezza per gli utenti o sui backup del sistema, può essere complesso.

Capitolo 4

Confronto soluzioni open source

Le soluzioni analizzate sono tutte open source, ad eccezione di vSAN, ma non sono tutte supportate allo stesso modo. I numeri su GitHub di Lustre e HDFS non sono particolarmente significativi, in quanto il codice di Lustre è hostato su un server proprietario, mentre HDFS fa parte della suite Hadoop, di conseguenza non è possibile ricavare le statistiche che riguardano solo il FS.

Comparazione soluzioni open source								
	Ceph	**GlusterFS**	**LizardFS**	**MinIO**	**OpenIO**	**HDFS**	**Lustre**	**SeaweedFS**
Stars	9,3k	3,2k	811	28,3k	488	11,7k	n.a.	12,2k
Watch	672	238	114	584	54	1k	n.a.	532
Commit	124k	16k	2k	8k	6k	25k	22k	5,8k
Fork	4,4k	919	170	3,1k	84	7,3k	n.a.	1,6k
Contributors	1083	235	43	298	31	364	116	120
Docker pull	50M+	5M+	50k+	500M+	10k+	10k+	1M+	1M+
Versione	16.2.4	9.3	3.13-rc3	2021-06-17T00-10-46Z	20.04	3.2.2	2.14.0	2,56
Ultimo aggiornamento	31/03/2021	28/06/2021	20/08/2020	17/06/2021	09/06/2020	09/01/2021	19/02/2021	28/06/2021
Anno inizio sviluppo	2007	2006	2013	2016	2006	2006	2007	2014
Partner	RedHat, Digital Ocean, Datadog, Runtastic, Samsung, Intel, Western Digital, ZTE	RedHat, Uline, Cisco, Koenig solutions, IBM	DIAWAY, Exertis, S3S, Syssoft	Intel, Dell, Nexus, Alibaba Travels, GitLab, Sendcloud, HP, Seagate	Elaia, Partech, OktoCampus	molto utilizzato anche dai principali social network fino a qualche anno fa, adesso quasi in disuso	Dell, HP, Fujitsu, Hitachi, NetApp, AWS (Amazon Fsx for Lustre)	Sugon Cloud

Figura 4.1. Comparazione soluzioni open source

Interesse

Analizzando i numeri su GitHub, le soluzioni con il minor numero di stars e watch sono LizardFS e OpenIO. Le soluzioni più apprezzate risultano essere invece Ceph, SeaweedFS

e in particolare MinIO, nonostante il suo sviluppo sia quello iniziato più recentemente. Lo strumento Google Trends restituisce uno scenario coerente con quanto visto su GitHub: come si evince dalla figura 4.2, l'interesse verso MinIO è in crescita ed ha recuperato un notevole gap nei confronti di Ceph, il quale è in discesa nel biennio 2020/2021. GlusterFS ha dei buoni numeri su GitHub, ma dall'analisi attraverso Google Trends si nota che l'interesse, che era sui livelli di Ceph, è in netto calo dal 2017. Per quanto riguarda HDFS, nonostane Hadoop sia un progetto in piena salute, si sta progressivamente perdendo interesse sul FS in favore di altre soluzioni, ed è comune trovare sviluppatori che chiedono se l'architettura sia ancora supportata.

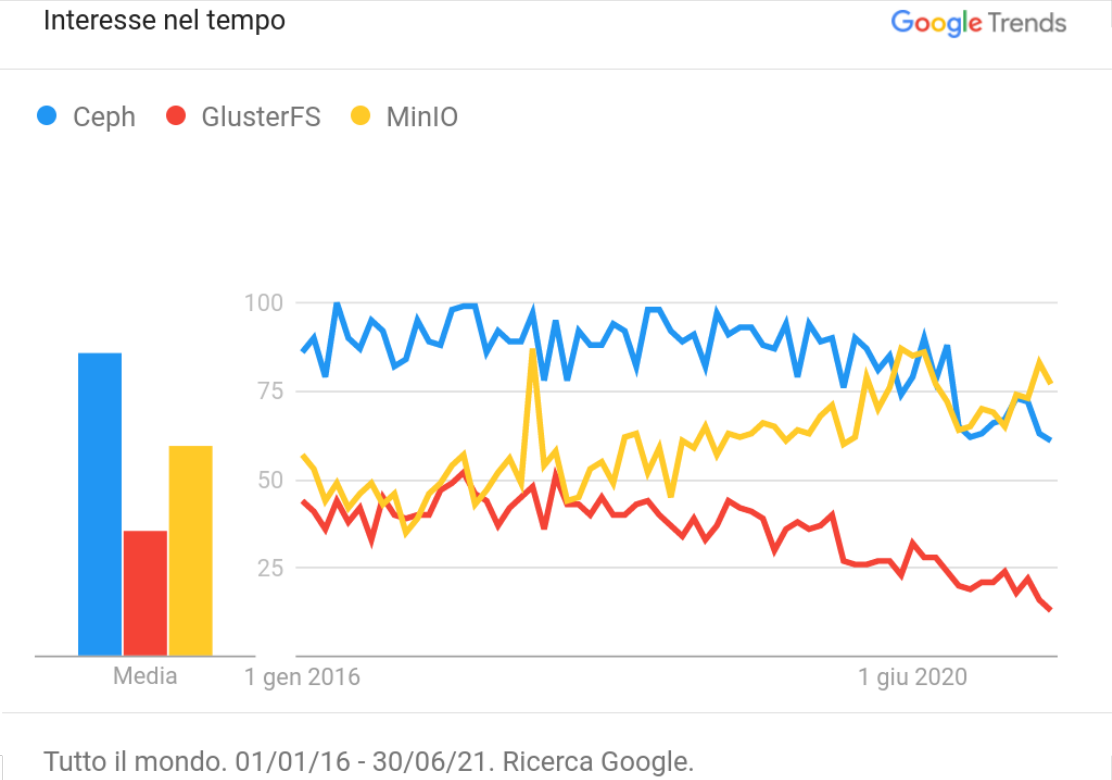

Figura 4.2. Andamento delle ricerche su Google dal 2016 al 2021

Sviluppo

Il numero di commit di Ceph è nettamente il più elevato, così come quello dei contributors e dei fork, evidenziando che sono molti gli sviluppatori che lavorano all'evoluzione del progetto. Il numero di commit di Gluster e di Lustre è superiore rispetto alle altre soluzioni, ma è rilevante il numero di fork di MinIO e SeaweedFS.

Aggiornamenti

Nonostante LizardFS sia la seconda soluzione più giovane dopo MinIO, il rilascio degli aggiornamenti è rallentato negli ultimi 3 anni, e l'ultimo update risale all'agosto 2020. Sul GitHub di OpenIO si trovano dei commit recenti, ma l'ultimo major update risale al giugno 2020. HDFS continua a ricevere aggiornamenti (ultimo nel gennaio 2021), Ceph rilascia bug fixes per tre versioni che vengono supportate per due anni dal rilascio, e una nuova versione viene rilasciata a cadenza annuale. MinIO e SeaweedFS sono le soluzioni con gli aggiornamenti più frequenti e danno molto supporto alla community.

Le nomenclature delle versioni variano molto da una soluzione all'altra, non è quindi possibile elaborare una comparativa sul rilascio dei major update a partire dall'inizio dello sviluppo.

Partner e diffusione

LizardFS, OpenIO e Lustre hanno dei partner che finanziano il progetto o, nel caso di Lustre, dei vendor che uniscono l'architettura di storage al proprio hardware, oltre ad avere un'architettura dedicata per Amazon AWS che sfrutta le sue alte prestazioni. Seppur Hadoop continui a essere molto utilizzato, HDFS non riscontra più successo secondo i riferimenti trovati.

SeaweedFS ha pochi partner, di conseguenza la sua diffusione in ambito produzione è difficilmente quantificabile.

Ceph e GlusterFS ricevono il supporto da RedHat, di conseguenza godono della presenza di molta documentazione, di tool e integrazione con architetture come OpenStack che vengono molto apprezzati dalla community.

MinIO, come detto in precedenza, è una soluzione che gode di molta attenzione soprattutto per essere cloud-native, ed è per questo supportata da grandi partner come Nexus, Dell, Intel.

I risultati raccolti sono confermati in termini di diffusione dal numero di pull da DockerHub: Ceph e MinIO risultano essere le soluzioni più diffuse con oltre 10 milioni di download ognuna, seguite da Gluster con oltre 5 milioni di download. Abbiamo poi Lustre e SeaweedFS con oltre 1 milione di download e le altre soluzioni con numero di download sotto il milione.

Osservazioni sulla comparativa

Dall'analisi emerge che Ceph e MinIO sono le soluzioni con le community più attive e con i principali partner alle spalle, e possono quindi essere considerate le architetture da utilizzare per uno storage privato o per un ambiente cloud.

In questa tesi, la soluzione che verrà analizzata in maniera maggiormente approfondita sarà Ceph, integrandolo in ambiente Kubernetes attraverso il progetto Rook. La decisione deriva dalla necessità di avere un file system e non solo uno storage a oggetti.

Parte III

Rook-Ceph

Capitolo 5

Rook

5.1 Presentazione del progetto

Rook è uno storage orchestrator open source che permette di integrare in ambiente cloud delle soluzioni di storage, automatizzando il deployment, la configurazione, il provisioning, lo scaling, il monitoring e la gestione delle risorse [3].

Il progetto viene incubato nel 2018 dalla Cloud Native Computing Foundation, per raggiungere lo stato di graduated project nell'ottobre 2020. L'idea alla base è quella di superare la necessità di gestire lo storage come uno strumento esterno, consentendo così al team IT che si occupa del cluster Kubernetes di avere anche il controllo sui dati persistenti. Infatti sono sempre di più le applicazioni stateful delle quali viene effettuato il deploy in container, e per questo motivo cresce il bisogno di automatizzare anche la configurazione dello storage all'interno del cluster, il quale è potenzialmente collegato a diversi servizi per la memorizzazione dei dati.

La coesistenza di più soluzioni di storage è garantita da Rook attraverso un Operator per ogni soluzione: in particolare, al momento della stesura della presente tesi, Rook supporta nativamente Ceph, Cassandra e NFS. Per via della maggiore diffusione, in questa tesi verrà analizzato solo il funzionamento di Ceph, essendo anche l'unica soluzione di storage supportata con stato "Stable" (il supporto a Cassandra e NFS è in stato "alpha" per entrambi).

5.2 Architettura e configurazione

5.2.1 Prerequisiti

Il cluster Kubernetes deve garantire che determinati requisiti vengano soddisfatti affinché Rook possa funzionare correttamente. In particolare Kubernetes deve essere aggiornato alla versione 1.11 o una più recente, inoltre è necessario che una delle seguenti condizioni sia verificata:

- un dispositivo senza partizioni o senza filesystem formattato

- una partizione senza filesystem formattato
- disponibilità di Persistent Volumes attraverso una Storage Class in modalità block

5.2.2 Operatore e componenti principali

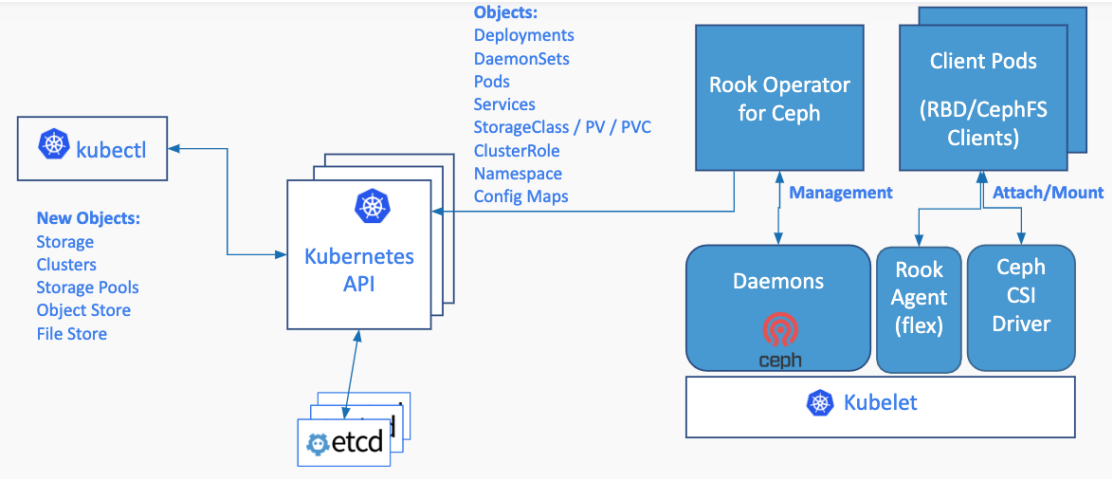

Figura 5.1. Architettura Rook-Ceph in Kubernetes [15]

Per effettuare il deploy di un cluster Rook-Ceph, è possibile utilizzare i file di esempio presenti nella repository GitHub di Rook [6]. In particolare, potrebbe essere sufficiente utilizzare i seguenti comandi:

```
git clone --single-branch --branch v1.6.7 https://github.com/rook/rook.git
cd rook/cluster/examples/kubernetes/ceph
kubectl create -f crds.yaml -f common.yaml -f operator.yaml
```

Si analizzano di seguito i tre file yaml utilizzati.

crds.yaml

Questo file contiene una serie di Custom Resource Definition (CRD) da creare prima di effettuare la creazione del vero e proprio cluster Rook-Ceph. Una CRD è un'estensione delle Kubernetes API che permette all'utente di creare le proprie risorse e interagirvi così come si interagisce con le risorse native di Kubernetes.

common.yaml

In questo file vengono definite delle risorse necessarie per il deploy dell'operatore e per la creazione del cluster. Il file d'esempio presente nel repository GitHub e utilizzato per la creazione dei cluster analizzati in questa tesi assume che verrà creato un operatore che

gestisce un singolo cluster nel namespace *rook-ceph*. Il namespace è infatti la prima risorsa creata da questo file, per proseguire poi con una serie di ClusterRole, ClusterRoleBinding, Role, RoleBinding e ServiceAccount per l'operatore e per i componenti del cluster Ceph (OSD, Mgr, CMD reporter) che definisce le possibilità di interazione tra i diversi pod e i permessi per l'accesso alle risorse

operator.yaml

L'operator è progettato per automatizzare la gestione dei componenti del cluster. Si occupa quindi di analizzare le proprietà del cluster definite come specifiche nella risorsa CephCluster e di effettuare il deploy di tutti i pod, i servizi e gli altri componenti che caratterizzano un cluster Ceph.

Il file contiene una serie di flag per personalizzare a basso livello l'interazione tra il cluster Ceph e Kubernetes attraverso la CSI.

5.2.3 Creazione del cluster Ceph

Il file *cluster.yaml* definisce la vera e propria configurazione del cluster Ceph. Tra le varie opzioni configurabili, ne vengono attenzionate alcune di seguito:

```
kind: CephCluster
metadata:
  name: rook-ceph
  namespace: rook-ceph
```

Viene così definito il nome usato per il cluster e il namespace creato nel file *common.yaml* che verrà utilizzato dall'operatore per creare le risorse.

```
spec:
  cephVersion:
    image: ceph/ceph:v16.2.4
```

È così possibile selezionare la versione di Ceph da utilizzare, specificando la versione dell'immagine da usare come sorgente.

```
  dataDirHostPath: /var/lib/rook
```

Questo path identifica dove verranno memorizzati sull'host i dati usati dai servizi. In caso di eliminazione del cluster, prima di poterne creare uno nuovo usando lo stesso path è necessario cancellare i dati contenuti in questa directory, altrimenti la creazione del nuovo cluster non andrà a buon fine. È molto importante attenzionare questo aspetto in quanto rende necessario che l'amministratore del cluster abbia accesso diretto ai nodi (ad esempio via ssh) per poter pulire il path nel caso in cui si abbia bisogno di ricreare il cluster.

```
...
  dashboard:
    enabled: true
    ssl: true
...
```

Ceph fornisce una comoda dashboard dalla quale è possibile tenere sotto controllo lo stato del cluster e dei singoli componenti, l'occupazione dello storage, il throughput, gli IOPS ecc.

Per essere raggiungibile dall'esterno del cluster però, è necessario esporre il servizio che viene creato dall'operator. Esistono diversi modi per esporre il servizio, due dei quali (Ingress Controller e Load Balancer) verranno esaminati nei prossimi capitoli.

Altri parametri come il numero di mon, mgr e la configurazione dello storage verranno analizzati nei prossimi capitoli in quanto maggiormente rilevanti nell'analisi dei diversi use cases.

5.3 Filesystem: configurazione e dettagli

Come analizzato in precedenza, uno dei principali vantaggi di Ceph è la possibilità di gestire con un unico servizio di storage sia storage a blocchi, sia storage a oggetti, sia filesystem condivisi.

In questa tesi, per effettuare i test di performance è stato utilizzato il filesystem condiviso, così da poter paragonare i risultati ottenuti utilizzando il filesystem locale con quelli ottenuti utilizzando un filesystem gestito da Rook-Ceph.

Sia on-premise, sia su Azure, i test sono stati effettuati creando un pod attraverso il file *direct-mount.yaml*, accedendovi alla bash e montando il filesystem condiviso con i seguenti comandi [2]:

```
mkdir /tmp/registry
mon_endpoints=$(grep mon_host /etc/ceph/ceph.conf | awk '{print $3}')
my_secret=$(grep key /etc/ceph/keyring | awk '{print $3}')
mount -t ceph -o mds_namespace=myfs,name=admin,secret=$my_secret \
$mon_endpoints:/ /tmp/registry
```

5.3.1 Filesystem replicato

I cluster utilizzati nei test prevedono la presenza di tre nodi di storage. Come conseguenza, il filesystem condiviso è stato configurato per avere una replica di file per ogni nodo. Questo garantisce la persistenza del dato anche nel caso in cui due dischi si danneggino.

Il flag *preserveFilesystemOnDelete* specifica se i dati devono essere conservati quando la risorsa CephFilesystem viene eliminata, evitando così perdite di dati in caso di cancellazione involontaria della risorsa.

Con metadataPool e dataPools si definiscono rispettivamente il numero di repliche dei pool di metadati e dei pool di dati.

Con metadataServer si definiscono le caratteristiche delle istanze MDS che dovranno essere create: in particolare con activeCount si definisce il numero di MDS, ricordando che CephFS ne creerà in automatico il doppio per sostituire le istanze attive specificate in activeCount in caso di failure; activeStanby è un flag che se settato a true fa in modo che le istanze MDS extra mantengano delle cache per ridurre il tempo di down in caso di failure.

Il file per la creazione del filesystem utilizzato nei test contiene quindi il seguente codice:

```
apiVersion: ceph.rook.io/v1
kind: CephFilesystem
metadata:
  name: myfs
  namespace: rook-ceph
spec:
  metadataPool:
    replicated:
      size: 3
  dataPools:
    - replicated:
        size: 3
  preserveFilesystemOnDelete: false
  metadataServer:
    activeCount: 1
    activeStandby: true
```

5.3.2 Filesystem con Erasure Code

Mentre con il filesystem replicato è richiesta un'occupazione dello spazio pari a N volte la dimensione del file, dove N è il numero di repliche, è invece possibile ottenere un risultato più efficiente in termini di memoria usando pool di tipo erasure code. Erasure code si basa sulla creazione di chunks di due diverse tipologie:

- data chunk: blocchi in cui il file viene diviso
- coding chunk: blocchi addizionali calcolati attraverso una funzione di codifica. Il numero di coding chunk identifica il numero di OSD che possono danneggiarsi senza avere perdita di dati

Nella figura 5.2 è mostrato un esempio di pool con erasure code: si ipotizzi di impostare il filesystem per la creazione di 4 data chunk e 2 coding chunk e di dover memorizzare un dato di dimensione 40 MB. Come primo passaggio il dato viene suddiviso in 4 chunk da 10 MB ognuno, e viene effettuata l'elaborazione dei 2 coding chunk, anch'essi di dimensione pari a 10 MB. Ogni chunk viene quindi memorizzato in un diverso nodo del cluster,

ipotizzando di avere un solo OSD per ogni nodo. Lo spazio occupato totale corrisponde quindi a circa 60 MB, garantendo la persistenza del dato anche con il failure di due nodi in contemporanea. Nel caso di filesystem replicato, per garantire la persistenza con lo stesso numero di failure contemporanei sarebbe necessario replicare l'intero dato 2 volte, per un'occupazione totale pari a 40 MB x 3 nodi = 120 MB.

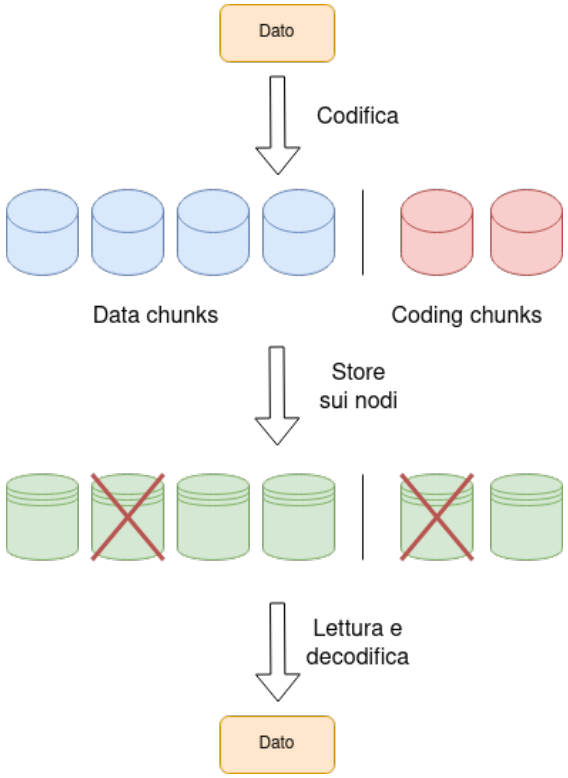

Figura 5.2. Esempio di filesystem con erasure code

Le operazioni di codifica e decodifica causano un rallentamento nelle performance dello storage, le quali verranno analizzate e confrontate nel dettaglio nei test riportati nei prossimi capitoli.

Il file yaml utilizzato per effettuare i test indica la creazione di 2 data chunk e 1 coding chunk, in quanto i cluster sono costituiti da 3 nodi di storage. Le altre opzioni presenti seguono lo schema utilizzato per il filesystem replicato visto nel paragrafo precedente.

```
apiVersion: ceph.rook.io/v1
kind: CephFilesystem
metadata:
  name: myfs
  namespace: rook-ceph
spec:
```

```
metadataPool:
  replicated:
    size: 3
dataPools:
  - erasureCoded:
      dataChunks: 2
      codingChunks: 1
metadataServer:
  activeCount: 1
  activeStandby: true
```

Capitolo 6

Deploy on-premise

I test on-premise sono stati effettuati su due diversi cluster: il primo cluster con nodi virtualizzati e storage basato su una SAN, il secondo cluster costituito da host fisici con dischi direttamente connessi.

6.1 Descrizione architettura

Il cluster Kubernetes basato sulla SAN è stato fornito dall'azienda Criticalcase ed è costituito da 8 nodi, dei quali 3 hanno ruolo di master node e 5 di worker node. Di questi 5 nodi, 3 hanno il ruolo di storage node e ospitano quindi gli OSD.

Ogni nodo è fornito di 2 vCPU e 8 GB di RAM, con Ubuntu 18.04.5 LTS come sistema operativo. La versione di Kubernetes che gestisce il cluster è la v1.20.6. La SAN utilizza sia SSD sia HDD: i primi sono dischi da 3.84TB, mentre gli HDD hanno una capacità di 8TB.

Il secondo cluster invece è composto da 6 server fisici situati presso il Politecnico di Torino. Tutti i nodi hanno ruolo di worker, con uno di essi che si occupa della gestione del control plane attraverso una VM KVM. Kubernetes è stato aggiornato alla versione 1.21, mentre sui nodi si ha Ubuntu 20.04 LTS.

Per caratteristiche, i nodi si dividono in due gruppi: 4 nodi hanno una CPU da 28 core logici, e sono equipaggiati con 256GB di RAM e SSD da 2TB, i restanti 2 nodi sono invece dotati di 2 CPU da 56 core logici, 512GB di RAM e SSD da 8 TB. L'infrastruttura di rete che collega i nodi prevede 2 connessioni 10Gbps in link aggregation.

6.2 Analisi file di deploy

I file di configurazione del cluster Rook-Ceph per il deploy on-premise sono prevalentemente i file originali presenti nel repository GitHub di Rook. La versione utilizzata per effettuare i test è la v1.6.5 rilasciata giorno 11 giugno 2021.

Entrando maggiormente nel dettaglio, sono stati usati senza modifiche i file *crds.yaml*, *common.yaml* e *operator.yaml*.

Per quanto riguardarda il deploy del cluster sui server dell'azienda è stato utilizzato il seguente codice:

```yaml
apiVersion: ceph.rook.io/v1
kind: CephCluster
metadata:
  name: rook-ceph
  namespace: rook-ceph
spec:
  cephVersion:
    image: ceph/ceph:v16.2.4
    allowUnsupported: false
  dataDirHostPath: /var/lib/rook
  skipUpgradeChecks: false
  continueUpgradeAfterChecksEvenIfNotHealthy: false
  waitTimeoutForHealthyOSDInMinutes: 10
  mon:
    count: 3
    allowMultiplePerNode: false
  mgr:
    count: 2
    modules:
      - name: pg_autoscaler
        enabled: true
  dashboard:
    enabled: true
    ssl: true
  monitoring:
    enabled: false
    rulesNamespace: rook-ceph
  network:
  crashCollector:
    disable: false
  cleanupPolicy:
    confirmation: ""
    sanitizeDisks:
      method: quick
      dataSource: zero
      iteration: 1
    allowUninstallWithVolumes: false
  annotations:
  labels:
  resources:
  removeOSDsIfOutAndSafeToRemove: false
  storage:
    useAllNodes: false
    useAllDevices: false
    config:
```

```
    nodes:
    - name: "storage-node-1"
      devices:
      - name: "sdb"
      config:
        storeType: bluestore
    - name: "storage-node-2"
      devices:
      - name: "sdb"
      config:
        storeType: bluestore
    - name: "storage-node-3"
      devices:
      - name: "sdb"
      config:
        storeType: bluestore
  disruptionManagement:
    managePodBudgets: true
    osdMaintenanceTimeout: 30
    pgHealthCheckTimeout: 0
    manageMachineDisruptionBudgets: false
    machineDisruptionBudgetNamespace: openshift-machine-api
  healthCheck:
    daemonHealth:
      mon:
        disabled: false
        interval: 45s
      osd:
        disabled: false
        interval: 60s
      status:
        disabled: false
        interval: 60s
    livenessProbe:
      mon:
        disabled: false
      mgr:
        disabled: false
      osd:
        disabled: false
```

Oltre a quanto analizzato nel paragrafo 5.2.3, con questo file vengono creati 3 mon e 2 mgr. Nella sezione `storage` si fa in modo che gli OSD vengano creati solamente nei nodi di storage: per far ciò, sono stati impostati a false i flag `useAllNodes` e `useAllDevices` e sono estati elencati nella sezione `nodes` i tre nodi di storage e i rispettivi device non formattati da utilizzare.

Il file utilizzato per il cluster del Politecnico usa la stessa configurazione, adattando solamente la sezione `storage` al diverso numero di nodi.

6.3 Test e prestazioni

Le performance dello storage sono state testate utilizzando Fio. In particolare, sono stati effettuati test di lettura e scrittura, prima con accesso sequenziale, poi con accesso randomico.

È stato creato un file di configurazione diviso in sezioni, così da poter avviare con ogni test una sezione diversa. Tra le varie impostazioni, ne vengono di seguito attenzionate alcune:

- directory: definisce la destinazione nella quale vengono scritti o letti i file per effettuare i test. È importante in quanto è l'unico campo che cambia nei test tra storage locale e storage distribuito con Ceph.
- blocksize: dimensione in byte dei blocchi per le unità di I/O. Ogni test è stato ripetuto più volte aumentando via via questo campo, per analizzare il diverso comportamento della soluzione di storage adottata al cambiare della dimensione dei blocchi.
- numjobs: numero di test Fio eseguiti in contemporanea. Per i test con accesso sequenziale questo campo è impostato a 1, mentre per i test ad accesso randomico è impostato a 8.
- runtime: durata del test, impostata sempre a 60 secondi.

```
[seq-read-8]
 # Sequential reads
 rw=read
 size=8g
 directory=/tmp/registry/utest
 fadvise_hint=0
 blocksize=8k
 direct=1
 buffered=0
 numjobs=1
 nrfiles=1
 runtime=60
 ioengine=libaio
 time_based
```

Per garantire la consistenza dei risultati, ogni test è stato eseguito 5 volte e viene rappresentata la media dei risultati, associando le corrispondenti barre di errore.

Benchmark accesso sequenziale su SAN

I primi test effettuati riguardano le performance in accesso sequenziale sull'architettura con SAN.

Nella figura 6.1 e nella figura 6.2 vengono messe a confronto la bandwidth con il numero di IOPS e la latenza media.

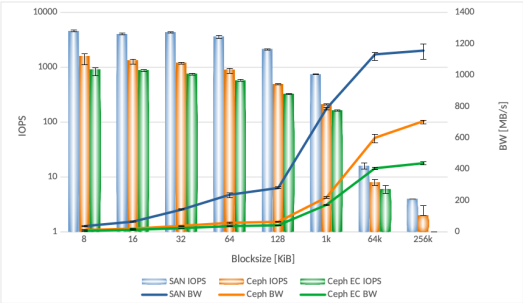

Figura 6.1. Benchmark IOPS - Bandwidth in lettura sequenziale su SAN

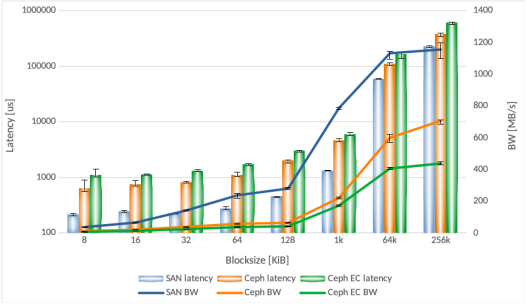

Figura 6.2. Benchmark latenza - Bandwidth in lettura sequenziale su SAN

Come da aspettative, le prestazioni ottenute utilizzando Ceph sono inferiori rispetto a quelle ottenute sulla SAN. In termini di bandwidth il gap non è così rilevante se si prendono in analisi dimensioni dei blocchi ridotte, ma diventa notevole al crescere della blocksize, con la BW sulla SAN che è circa due volte maggiore della velocità raggiunta su filesystem replicato e quasi tre volte maggiore delle performance raggiunte con Erasure Code. I risultati sul numero di IOPS e sulla latenza mantengono le stesse proporzioni.

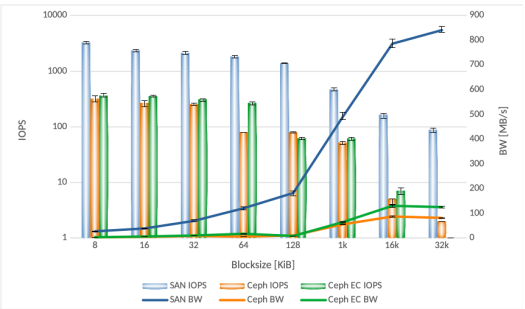

Figura 6.3. Benchmark IOPS - Bandwidth in scrittura sequenziale su SAN

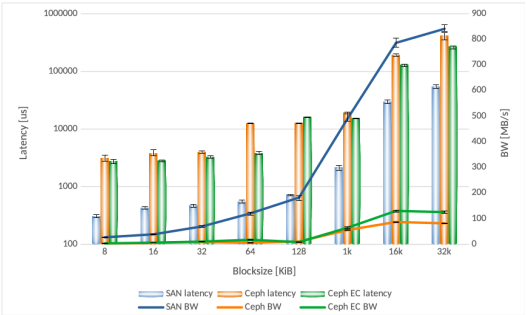

Figura 6.4. Benchmark latenza - Bandwidth in scrittura sequenziale su SAN

I grafici 6.3 e 6.4 rappresentano i risultati dei test di scrittura sequenziale su SAN. Le prestazioni raggiunte scrivendo sui dischi sono generalmente inferiori rispetto a quelle raggiunte in lettura, ma è interessante notare come si sia ampliata la differenza rispetto alle performance raggiunte tramite Ceph.

La principale differenza dai test in lettura è però riscontrabile paragonando il comportamento di Ceph con le due diverse tipologie di filesystem. Mentre in lettura le performance

con il filesystem replicato sono sempre migliori, in scrittura il filesystem con Erasure Code riesce a garantire una minore latenza, maggior numero di IOPS e maggiore velocità soprattutto al crescere della blocksize.

Benchmark accesso sequenziale su storage direttamente connesso

I successivi test sono stati effettuati sull'infrastruttura con dischi direttamente connessi ai nodi facendone accesso sequenziale in lettura e in scrittura.

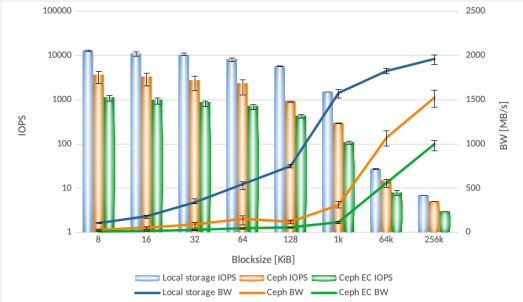

Figura 6.5. Benchmark IOPS - Bandwidth in lettura sequenziale su local storage

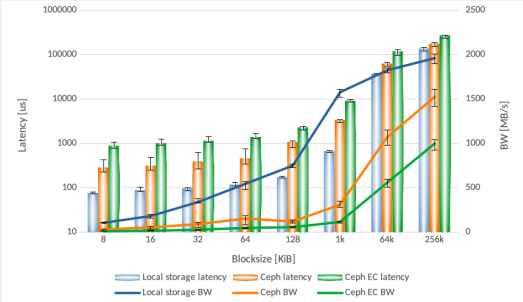

Figura 6.6. Benchmark latenza - Bandwidth in lettura sequenziale su local storage

I grafici 6.5 e 6.6 mostrano delle performance migliori rispetto a quelle ottenute nei test in lettura sequenziale sulla SAN mostrati in precedenza, sia nell'accesso diretto, sia con Ceph. Anche la differenza tra le diverse soluzioni è ridotta, con la banda raggiunta accedendo al disco locale pari a circa 1.5 volte la banda raggiunta con Ceph configurato con filesystem replicato e circa 2 volte la banda raggiunta usando Erasure Code.

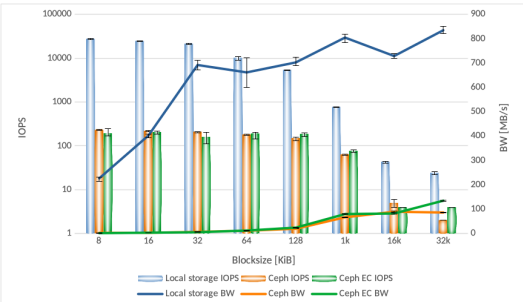

Figura 6.7. Benchmark IOPS - Bandwidth in scrittura sequenziale su local storage

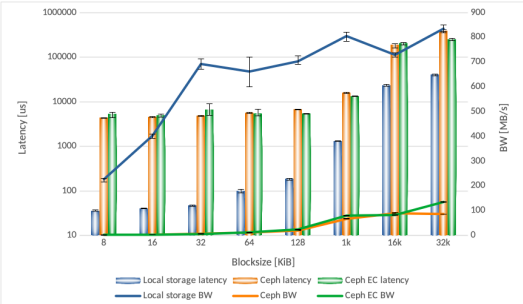

Figura 6.8. Benchmark latenza - Bandwidth in scrittura sequenziale su local storage

I risultati ottenuti in scrittura mostrati nelle figure 6.7 e 6.8 invece sono molto simili in termini di picchi raggiunti rispetto all'infrastruttura con SAN, ma in questo contesto

il filesystem con Erasure Code raggiunge delle performance molto simili al filesystem replicato, riuscendo a creare un distacco rilevante solo con blocchi di dimensione 32MiB.

Benchmark accesso randomico su SAN

I test ad accesso randomico prevedono l'esecuzione di 8 jobs in parallelo. I risultati rappresentati sono restituiti da Fio come cumulativi per l'intero test.

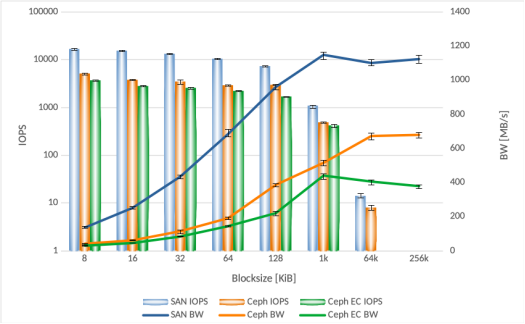

Figura 6.9. Benchmark IOPS - Bandwidth in lettura randomica su SAN

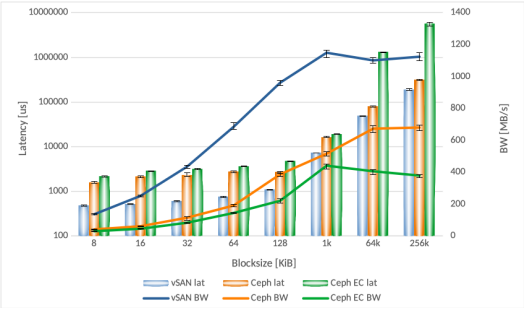

Figura 6.10. Benchmark latenza - Bandwidth in lettura randomica su SAN

In termini di bandwidth, i grafici 6.9 e 6.10 mostrano delle velocità massime simili a quelle raggiunte in accesso sequenziale, sia accedendo ai dischi sia con Ceph, mentre si ottengono delle performance migliori sulle blocksize inferiori.

Il numero di IOPS ottenuti è invece inferiore, così come è superiore la latenza media ottenuta.

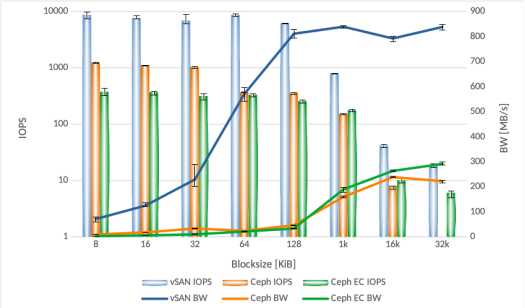

Figura 6.11. Benchmark IOPS - Bandwidth in scrittura randomica su SAN

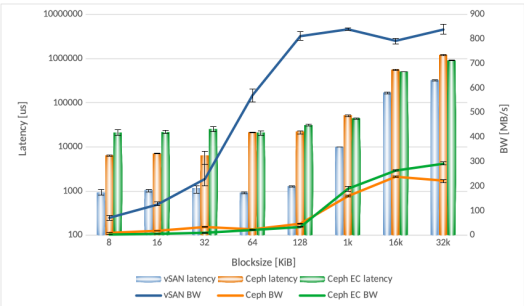

Figura 6.12. Benchmark latenza - Bandwidth in scrittura randomica su SAN

Anche i risultati nei test in scrittura randomica rappresentati nei grafici 6.9 e 6.10 hanno le stesse caratteristiche dei test in lettura nella comparativa con l'accesso sequenziale,

ovvero velocità di scrittura maggiori per dimensioni dei blocchi minori ma si ottengono performance generalmente peggiori negli IOPS e nella latenza.

Anche nell'accesso randomico, si nota come il filesystem con Erasure Code permetta di ottenere prestazioni migliori rispetto al filesystem replicato.

Benchmark accesso randomico su storage direttamente connesso

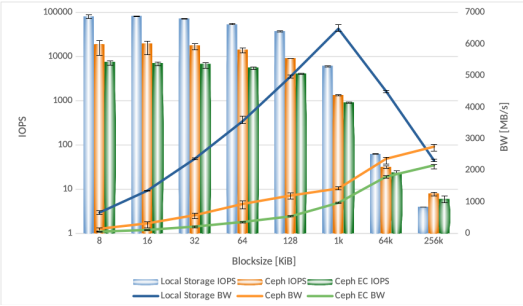

Figura 6.13. Benchmark IOPS - Bandwidth in lettura randomica su local storage

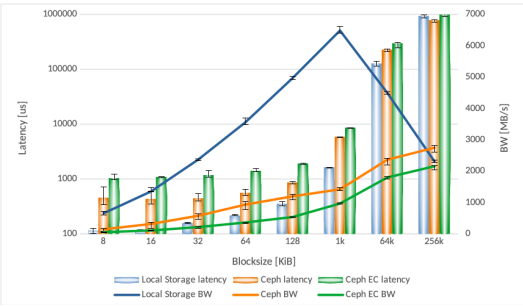

Figura 6.14. Benchmark latenza - Bandwidth in lettura randomica su local storage

Nei test effettuati in lettura randomica, come visibile in figura 6.13 e 6.14 la bandwidth risulta essere molto maggiore rispetto all'accesso sequenziale con tutte le soluzioni analizzate. Per dimensioni dei blocchi maggiori di 1MB le prestazioni in velocità vedono una netta diminuzione. Tale peggioramento è intrinseco nell'architettura, in quanto per verificare le cause sono stati effettuati dei test equivalenti anche su bare metal senza passare dalla virtualizzazione e i risultati sono coerenti con quanto ottenuto sul pod di test.

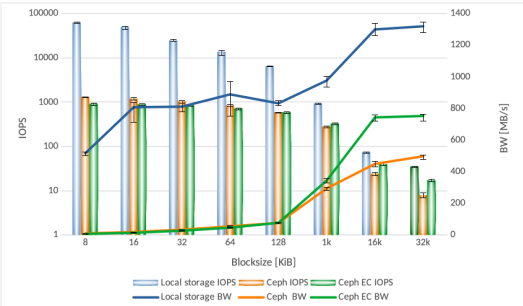

Figura 6.15. Benchmark IOPS - Bandwidth in scrittura randomica su local storage

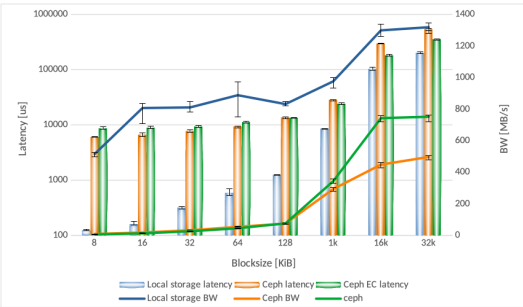

Figura 6.16. Benchmark latenza - Bandwidth in scrittura randomica su local storage

Anche nei test in scrittura le prestazioni risultano maggiori. È visibile anche in questo scenario il miglior comportamento di Ceph con Erasure Code per blocksize maggiore di 128KiB.

Capitolo 7

Azure

7.1 Descrizione architettura

Per creare un cluster in una soluzione di cloud pubblico è stato utilizzato Azure Kubernetes Service (AKS).

Il primo passaggio è creare un resource group che conterrà il servizio Kubernetes, per poi procedere con la configurazione delle risorse. Vengono creati due pool di nodi composti da 3 nodi ognuno: il primo per gestire il carico di lavoro di eventuali applicazioni, il secondo dedicato invece allo storage. È possibile quindi selezionare la versione di Kubernetes con cui creare il cluster, per i seguenti test la v1.20.7, e la dimensione dei nodi, in questo caso la DS2_v2 caratterizzata da 2 vCPU, 7 GiB di RAM e 6400 IOPS massimi.

A causa della creazione di due pool di nodi, è necessario usare i taint per fare in modo che non vengano creati pod nei nodi di storage se non permesso esplicitamente. Il comando per aggiungere i taint è:

```
kubectl taint nodes node1 storage-node=true:NoSchedule
```

Pool di nodi	Stato	Numero di nodi	Modalità	Versione di Kubernetes	Dimensioni del nodo	Sistema operativo
npstandard	In esecuzione	3/3 pronto	Sistema	1.20.7	Standard_DS2_v2	Linux
npstorage	In esecuzione	3/3 pronto	Utente	1.20.7	Standard_DS2_v2	Linux

Figura 7.1. Pool di nodi dalla dashboard Azure

7.2 Analisi file di deploy

Il cluster viene creato con la v1.6.5 di Rook, partendo dai file *crds.yaml* e *common.yaml* presenti nella directory del repository GitHub di Rook.

Per quanto riguarda il deploy dell'operator, non sono state apportate modifiche al file di esempio *operator.yaml*, ad eccezione delle seguenti righe di codice

```
spec:
...
    spec:
    ...
      containers:
      - name: rook-ceph-operator
        ...
        env:
        ...
        - name: CSI_PROVISIONER_TOLERATIONS
          value: |
            - effect: NoSchedule
              key: storage-node
              operator: Exists
        - name: CSI_PLUGIN_TOLERATIONS
          value: |
            - effect: NoSchedule
              key: storage-node
              operator: Exists
...
```

necessarie per via dei taint, così che i pod di Rook vengano schedulati nei nodi di storage.

Il cluster è composto da 3 OSD da 30 GiB l'uno e 3 mon. Le principali differenze con il file di esempio *cluster.yaml* consistono nello specificare la storage class *managed-premium* per fare in modo che vengano utilizzati gli SSD invece degli HDD, e le righe di codice per assegnare un OSD per ogni nodo di storage. Di seguito il codice usato per creare il cluster:

```
apiVersion: ceph.rook.io/v1
kind: CephCluster
metadata:
  name: rook-ceph
  namespace: rook-ceph
spec:
  dataDirHostPath: /var/lib/rook
  mon:
    count: 3
    allowMultiplePerNode: false
  cephVersion:
    image: ceph/ceph:v16.2.4
    allowUnsupported: false
  dashboard:
    enabled: true
    ssl: true
```

```
network:
  hostNetwork: false
storage:
  storageClassDeviceSets:
  - name: set1
    count: 3
    portable: true
    placement:
      tolerations:
      - key: storage-node
        operator: Exists
      nodeAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
          nodeSelectorTerms:
          - matchExpressions:
            - key: agentpool
              operator: In
              values:
              - npstorage
      podAntiAffinity:
        preferredDuringSchedulingIgnoredDuringExecution:
        - weight: 100
          podAffinityTerm:
            labelSelector:
              matchExpressions:
              - key: app
                operator: In
                values:
                - rook-ceph-osd
              - key: app
                operator: In
                values:
                - rook-ceph-osd-prepare
            topologyKey: kubernetes.io/hostname
    volumeClaimTemplates:
    - metadata:
        name: data
      spec:
        resources:
          requests:
            storage: 30Gi
        storageClassName: managed-premium
        volumeMode: Block
        accessModes:
          - ReadWriteOnce
disruptionManagement:
  managePodBudgets: false
  osdMaintenanceTimeout: 30
  manageMachineDisruptionBudgets: false
```

```
    machineDisruptionBudgetNamespace: openshift-machine-api
```

```
salvatore@Azure:~$ kubectl -n rook-ceph get pods
NAME                                                     READY   STATUS      RESTARTS   AGE
csi-cephfsplugin-5zfcc                                   3/3     Running     0          3d
csi-cephfsplugin-6l2r2                                   3/3     Running     0          3d
csi-cephfsplugin-8m79m                                   3/3     Running     0          3d
csi-cephfsplugin-dqsn4                                   3/3     Running     0          3d
csi-cephfsplugin-m2x48                                   3/3     Running     0          3d
csi-cephfsplugin-provisioner-7f7c95d6d-5czth             6/6     Running     0          3d
csi-cephfsplugin-provisioner-7f7c95d6d-swkzf             6/6     Running     0          3d
csi-cephfsplugin-rw22c                                   3/3     Running     0          3d
csi-rbdplugin-44vgw                                      3/3     Running     0          3d
csi-rbdplugin-bggg2                                      3/3     Running     0          3d
csi-rbdplugin-provisioner-566cbdf89c-2h2qz               6/6     Running     0          3d
csi-rbdplugin-provisioner-566cbdf89c-67qjm               6/6     Running     0          3d
csi-rbdplugin-rnmm4                                      3/3     Running     0          3d
csi-rbdplugin-vgxrx                                      3/3     Running     0          3d
csi-rbdplugin-z8j9v                                      3/3     Running     0          3d
csi-rbdplugin-znfkt                                      3/3     Running     0          3d
rook-ceph-crashcollector-aks-npstandard-23257502-vmss000005j9cq  1/1     Running     0          3d
rook-ceph-crashcollector-aks-npstandard-23257502-vmss000008s5pb  1/1     Running     0          3d
rook-ceph-crashcollector-aks-npstandard-23257502-vmss00000ntmsk  1/1     Running     0          3d
rook-ceph-crashcollector-aks-npstorage-23257502-vmss000000jmczq  1/1     Running     0          3d
rook-ceph-crashcollector-aks-npstorage-23257502-vmss000001qjh2p  1/1     Running     0          3d
rook-ceph-crashcollector-aks-npstorage-23257502-vmss000002rqmlj  1/1     Running     0          3d
rook-ceph-mds-myfs-a-7dd9b7996-zz8kv                     1/1     Running     0          19h
rook-ceph-mds-myfs-b-bcb78fdd6-fs7cp                     1/1     Running     0          19h
rook-ceph-mds-myfs-ec-a-7f4b6fcc85-wl899                 1/1     Running     0          21h
rook-ceph-mds-myfs-ec-b-6458689f7c-764jh                 1/1     Running     0          21h
rook-ceph-mgr-a-67f689d7cc-hsxv2                         1/1     Running     0          3d
rook-ceph-mon-a-5689c989db-wv7k7                         1/1     Running     0          3d
rook-ceph-mon-b-6c9846f587-z9r5v                         1/1     Running     0          3d
rook-ceph-mon-c-78c5d9fbb8-6xr7x                         1/1     Running     6          3d
rook-ceph-operator-778ffc6cbc-cj49k                      1/1     Running     0          3d
rook-ceph-osd-0-6f5c77bf8b-2wdwv                         1/1     Running     0          3d
rook-ceph-osd-1-66886c8f54-25d9h                         1/1     Running     0          3d
rook-ceph-osd-2-67df47dc75-2fx6j                         1/1     Running     0          3d
rook-ceph-osd-prepare-set1-data-074ssm-vvdj7             0/1     Completed   0          3d
rook-ceph-osd-prepare-set1-data-1gmthx-kgzfl             0/1     Completed   0          3d
rook-ceph-osd-prepare-set1-data-2nq86k-bskq9             0/1     Completed   0          3d
rook-ceph-tools-78cdfd976c-cdrtk                         1/1     Running     0          3d
rook-direct-mount-7978585f65-7gc79                       1/1     Running     0          19h
```

Figura 7.2. Pod del cluster su AKS

7.3 Test e prestazioni

La struttura dei file di configurazione di Fio usati per effettuare i test è equivalente a quella presentata nel paragrafo 6.3, sono quindi stati effettuati test di performance sui dischi premium Azure e sui filesystem gestiti da Ceph, sia in modalità replica sia con erasure code, aumentando via via la dimensione dei blocchi.

Benchmark accesso sequenziale

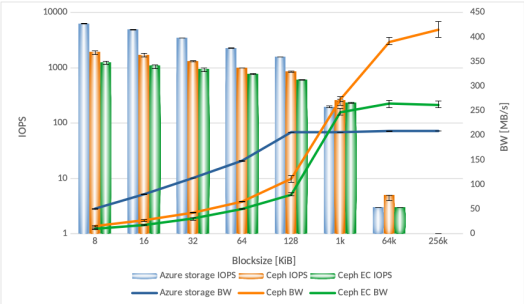

Figura 7.3. Benchmark IOPS - Bandwidth in lettura sequenziale su dischi Azure

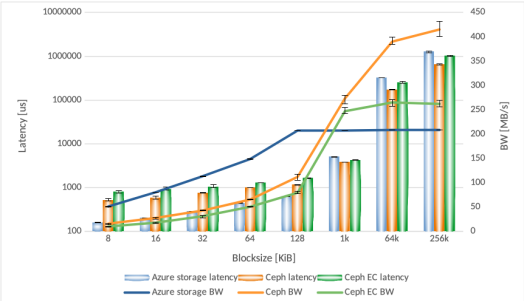

Figura 7.4. Benchmark latenza - Bandwidth in lettura sequenziale su dischi Azure

I test in lettura sequenziale mostrati nei grafici 7.3 e 7.4 mostrano un comportamento presente anche nei test successivi, sia in lettura sia in scrittura, sia con accesso sequenziale sia randomico, per quanto riguarda l'accesso ai dischi azure senza utilizzare Ceph: superata una determinata blocksize, diversa a seconda del test, viene raggiunto il limite di velocità specifico della dimensione del nodo selezionata.

Utilizzando la soluzione di storage distribuito invece tale limite viene superato, rendendo quindi anche le performance un vantaggio dell'utilizzo di Ceph su Cloud pubblico per grandi dimensioni dei blocchi.

Nel test di lettura sequenziale la blocksize oltre la quale si raggiunge la velocità massima è 128KB e dalle dimensioni superiori le performance di Ceph con filesystem replicato e filesystem con Erasure Code diventano superiori, sia in termini di bandwidth sia in termini di IOPS e latenza.

Figura 7.5. Benchmark IOPS - Bandwidth in scrittura sequenziale su dischi Azure

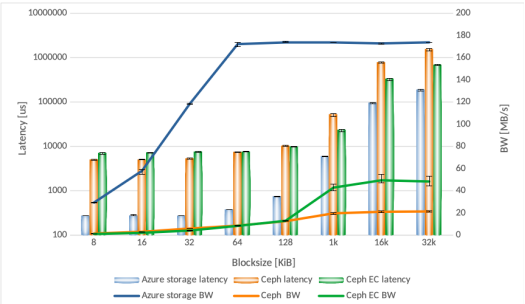

Figura 7.6. Benchmark latenza - Bandwidth in scrittura sequenziale su dischi Azure

In scrittura Ceph non riesce a offrire delle prestazioni sufficientemente elevate da raggiungere come in lettura le prestazioni dell'accesso ai dischi Azure una volta raggiunta la

soglia, come osservabile nei grafici 7.5 e 7.6. In questo scenario infatti la velocità raggiunta dalla soluzione di storage distribuito è notevolmente più bassa, così come il numero di IOPS, e la latenza calcolata risulta essere un ordine di grandezza superiore per blocksize elevata e due ordini di grandezza in caso di blocksize più basse.

Per dimensioni dei blocchi superiori a 128KiB il filesystem con Erasure Code riesce a ottenere delle velocità circa 2 volte superiori al filesystem replicato.

Benchmark accesso randomico

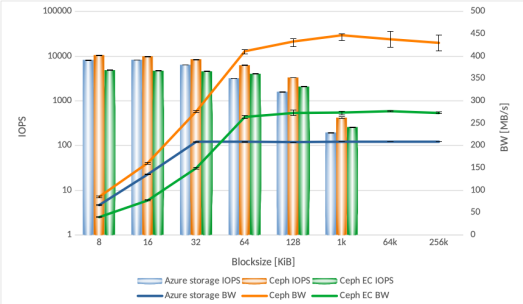

Figura 7.7. Benchmark IOPS - Bandwidth in lettura randomica su dischi Azure

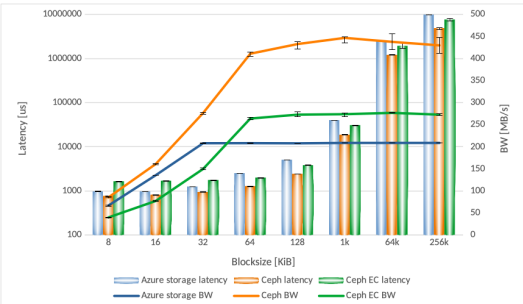

Figura 7.8. Benchmark latenza - Bandwidth in lettura randomica su dischi Azure

I grafici 7.7 e 7.8 mostrano che in lettura randomica il limite sulla bandwidth viene raggiunto con una blocksize di 32KiB, con delle latenze molto più elevate rispetto all'accesso sequenziale.

Questo scenario risulta essere l'unico tra quelli presi in analisi nel quale le prestazioni di Ceph, in particolare con filesystem replicato, risultano essere sempre migliori rispetto all'accesso diretto, anche con dimensioni dei blocchi ridotte. Utilizzando il filesystem con Erasure Code di Ceph le prestazioni superano la soglia massima della dimensione del nodo nei test con dimensione dei blocchi maggiore di 64KiB, raggiungendo una bandwidth che diventa all'incirca costante.

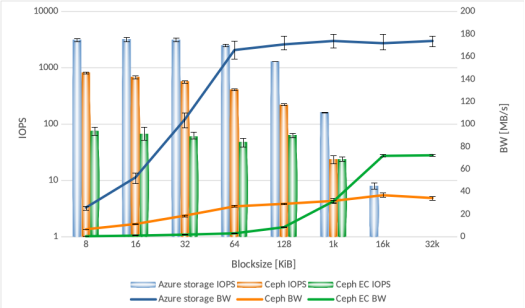

Figura 7.9. Benchmark IOPS - Bandwidth in scrittura randomica su dischi Azure

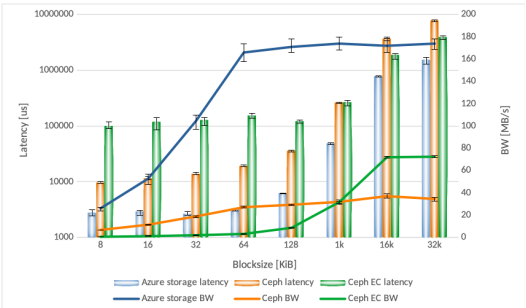

Figura 7.10. Benchmark latenza - Bandwidth in scrittura randomica su dischi Azure

L'ultimo scenario preso in analisi e raffigurato nei grafici 7.9 e 7.10 riguarda la scrittura in accesso randomico. Così come per l'accesso sequenziale le performance della soluzione di storage distribuito non raggiungono le prestazioni con accesso diretto ai dischi premium di Azure. Con questa configurazione i test sui dischi Azure mostrano lo stabilizzarsi della bandwidth a partire da blocchi di dimensione 64KiB e il filesystem con Erasure Code supera le performance del filesystem replicato per blocchi di dimensione maggiore di 1MiB.

Capitolo 8

Confronto risultati

I test analizzati negli scorsi capitoli mostrano che Rook-Ceph è una valida soluzione per l'astrazione dello storage distribuito, la quale permette in maniera semplice, attraverso poche modifiche nei file di configurazione, di gestire i dati svincolandosi dall'hardware sottostante.

L'aver effettuato i test sia on-premise sia su cloud pubblico, in particolare con due diverse architetture per quanto riguarda l'on-premise, ha permesso di osservare delle caratteristiche del comportamento di Ceph, alcune delle quali comuni a tutte le architetture. I due principali aspetti da sottolineare riguardano il comportamento dei filesystem gestiti da Ceph nei test in lettura su Azure e nei test in scrittura su tutte le configurazioni.

Per quanto riguarda i test in lettura su Azure, si ottengono performance migliori utilizzando Ceph rispetto all'accesso diretto ai dischi Azure per grandi dimensioni dei blocchi in quanto viene superata la soglia di bandwidth massima del nodo. Questo risultato è rilevante in quanto la perdita di prestazione è uno dei principali svantaggi nell'utilizzo di una soluzione di storage distribuito, di conseguenza avere la possibilità di garantire le proprietà di quest'ultima, tra le quali in particolare la ridondanza dei dati, ottenendo anche performance migliori la rende, per questo specifico scenario, la soluzione preferibile sotto tutti gli aspetti.

L'altro aspetto importante consiste nei migliori risultati ottenuti in scrittura utilizzando il filesystem con Erasure Code di Ceph piuttosto che il filesystem replicato. Attraverso Prometheus e Grafana durante i test sono stati osservati gli utilizzi di CPU e RAM nei diversi scenari, in particolare sul cluster on-premise con i dischi direttamente connessi ai nodi, analizzando i consumi generati dai pod appartenenti al namespace rook-ceph. Si è quindi visto che con questa architettura sottostante e per questa tipologia di test in scrittura, le differenze di consumi generati dai due diversi filesystem sono irrilevanti. Tale osservazione incrementa il numero dei vantaggi nell'utilizzo del filesystem con Erasure Code in quanto è possibile ottenere la sopravvivenza dei dati a un numero di failure di nodi pari a quello del filesystem replicato ma con una minore occupazione di spazio, senza un aumento visibile nell'utilizzo delle risorse e con performance uguali e a volte maggiori.

Parte IV

Conclusioni

Capitolo 9

Conclusioni generali

Il mondo del cloud computing è in continua evoluzione, il numero di applicazioni che richiedono elaborazione su server remoti continua ad aumentare, sia per quanto riguarda le aziende sia per quanto riguarda gli utenti finali, e di conseguenza aumenta anche il numero di dati da gestire e la complessità nel farlo, in quanto bisogna garantire in maniera efficiente la disponibilità dei dati e la loro sicurezza.

Nell'ambito cloud inoltre vi è la necessità di semplificare e automatizzare la gestione dei server e degli applicativi che vi girano sopra, rendendo quindi obbligatorio trovare una soluzione che possa permettere di avere accesso a ogni parametro di configurazione senza il bisogno di interagire con ogni componente dell'architettura, ma attraverso uno strato software che permetta di astrarre tutto l'hardware. L'utilizzo della virtualizzazione e di orchestratori come Kubernetes ha permesso di superare questo ostacolo e di creare cluster con nodi situati anche in parti diverse del mondo.

La possibilità di gestire in maniera centralizzata hardware collocato in aree diverse viene applicata anche all'ambito storage attraverso le soluzioni di storage distribuito, affinché si possa facilitare l'accesso ai dati recuperandoli dal server più vicino e garantendone l'integrità attraverso la replica o tecnologie come l'Erasure Code. Le soluzioni esistenti sono varie, sia open source sia closed, e sono in continuo sviluppo. Ogni soluzione si concentra su una o più caratteristiche, come possono essere ad esempio le performance, la sicurezza dei dati o la compatibilità con il maggior numero di sistemi possibili. Alcune di queste soluzioni stanno perdendo attrattiva, seppur in maniera relativa, si vedano ad esempio HDFS e GlusterFS, mentre altre soluzioni ottengono sempre più apprezzamenti, come MinIO e Ceph. Mentre MinIO supporta solo storage a oggetti, Ceph permette di configurare anche storage a blocchi e filesystem. Per questa maggiore flessibilità, la soluzione che è stata presa in analisi è Ceph.

In particolare, per poter utilizzare Ceph con Kubernetes, è stato utilizzato lo storage orchestrator Rook, un progetto open source che mette a disposizione un operatore per eseguire il deploy e la gestione di un cluster Ceph, personalizzabile attraverso la modifica di un file yaml di configurazione. Attraverso la modifica di questi parametri è stato infatti possibile effettuare il deploy di Ceph su due architetture on-premise, differenziate dall'utilizzo di una SAN o di dischi direttamente connessi, e su un cluster su cloud pubblico, in particolare Microsoft Azure.

Per verificare il comportamento di Rook-Ceph è stato utilizzato il tool open-source Fio, attraverso il quale sono stati effettuati dei test in lettura e scrittura, prima con accesso sequenziale poi randomico, aumentando progressivamente la dimensione dei blocchi. I risultati ottenuti mostrano come Rook-Ceph sia una valida e affidabile soluzione per astrarre la gestione dello storage distribuito, in quanto il comportamento mostrato è risultato coerente tra tutte le diverse architetture, tralasciando le dovute differenze prestazionali dovute ai limiti fisici dell'hardware sottostante. In certi scenari, come ad esempio la lettura sequenziale su cloud pubblico, l'utilizzo di Ceph ha permesso di ottenere prestazioni migliori rispetto all'utilizzo diretto dei dischi Azure garantendo in più i vantaggi dello storage distribuito, e in altre situazioni, come la scrittura sia sequenziale sia randomica per blocksize elevata, l'utilizzo di filesystem con Erasure Code ha garantito performance maggiori del filesystem replicato, oltre al minore spazio occupato per garantire la resilienza del dato essendo una caratteristica di tale tecnologia. In termini di utilizzo di CPU e RAM inoltre, i test effettuati non evidenziano differenze rilevanti tra filesystem replicato e filesystem con Erasure Code, di conseguenza la scelta dipende dallo scenario nel quale verrà utilizzato il filesystem e nella quantità di storage a disposizione.

In conclusione, le soluzioni di storage distribuito sono numerose e si prestano a diverse tipologie di utilizzo a seconda delle caratteristiche che si reputano necessarie. Ceph, grazie all'integrazione con Rook, si è rivelata una soluzione relativamente semplice da configurare e con dei comportamenti coerenti su tutte le architetture, con una community molto attiva che continua a svilupparla e che offre supporto per la risoluzione dei problemi che possono comparire quando si effettua una personalizzazione più profonda del cluster.

Capitolo 10

Sviluppi futuri

Le soluzioni di storage distribuito, così come le tecnologie di virtualizzazione e orchestrazione, sono in continuo sviluppo e offrono via via nuove funzionalità e servizi. È però non corretto valutare la validità di una soluzione solo analizzando le performance, in quanto sono numerose le caratteristiche da prendere in considerazione durante un progetto per selezionare la soluzione migliore: costi, supporto, tipologia di storage, compatibilità con l'hardware e sistemi operativi, possibilità di accesso ai dati sono tutti aspetti da valutare.

Uno sviluppo futuro può consistere nell'osservazione del funzionamento di Rook-Ceph e/o di altre soluzioni con degli scenari d'uso reale, ad esempio verificando il comportamento in termini di reattività e affidabilità di un'applicazione che utilizzi un filesystem gestito dalla soluzione di storage distribuito on-premise e su cloud pubblico.

Per avere una visione più completa, si potrebbe anche comparare il comportamento di più soluzioni di storage utilizzando tool più completi del solo Fio, come il benchmark CNSBench [11] presentato nel febbraio 2021.

Bibliografia

[1] *HDFS Architecture*. Apache Software Foundation, 2021. URL https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html.

[2] Rook authors. Shared filesystem tools, rook documentation, 2021. URL https://rook.io/docs/rook/v1.6/direct-tools.html#shared-filesystem-tools.

[3] Rook authors. *Rook Documentation*, 2021. URL https://rook.io/docs/rook/v1.6/.

[4] Jens Axboe. *Fio*, 2020. URL https://github.com/axboe/fio.

[5] Monu Bambroo. *AKS Series – Use Azure Storage Option as Persistent Volumes in AKS*, 2019. URL https://blogmonu.azurewebsites.net/?p=1328.

[6] Rook dev community. Repository github del progetto rook-ceph, 2021. URL https://github.com/rook/rook/blob/master/cluster/examples/kubernetes/ceph.

[7] *Gluster Docs - Architecture*. Gluster.org, 2021. URL https://docs.gluster.org/en/latest/Quick-Start-Guide/Architecture/.

[8] *about*. LizardFS, 2021. URL https://lizardfs.com/about.

[9] Chris Lu. *SeaweedFS wiki*, 2021. URL https://github.com/chrislusf/seaweedfs/wiki.

[10] Malcolm. *Introduction to Lustre*, 2017. URL https://wiki.lustre.org/Introduction_to_Lustre.

[11] Alex Merenstein, Vasily Tarasov, Ali Anwar, Deepavali Bhagwat, Julie Lee, Lukas Rupprecht, Dimitris Skourtis, Yang Yang, and Erez Zadok. *CNSBench: A Cloud Native Storage Benchmark*, 2021. URL https://www.usenix.org/system/files/fast21-merenstein.pdf.

[12] *MinIO architecture*. MinIO Inc, 2020. URL https://min.io/product/overview#.

[13] *Our Experiences with Ceph - Part 1*. nine Team, 2016. URL https://www.nine.ch/en/engineering-logbook/our-experiences-with-ceph-1.

[14] *Data Access*. OpenIO dev community, 2020. URL https://min.io/product/overview#.

[15] Mike Vizards. *CNCF Graduates Rook to Automate Kubernetes Storage Tasks*, 2020. URL https://containerjournal.com/topics/container-ecosystems/cncf-graduates-rook-to-automate-kubernetes-storage-tasks/.

[16] *vSAN7 Technology Overview - Architecture*. VMware, 2020. URL https://core.vmware.com/resource/vsan-7-technology-overview.