Ceph Technology Know-How

Daniele Schiavi, Nicola Atti

Indice

- Introduzione
- 1. Architettura e funzionamento di Ceph
  - 1.1. Panoramica generale di Ceph
  - 1.2. Panoramica tecnica di Ceph
  - 1.3. Componenti dello storage cluster Ceph
  - 1.3.1. Manager
  - 1.3.2. Monitor
  - 1.3.3. Metadata Server
  - 1.3.4. Object Storage Daemon
  - 1.4. RADOS ed il suo funzionamento
  - 1.4.1. Posizionamento dei dati
  - 1.4.2. CRUSH (Controlled Replication Under Scalable Hashing)
  - 1.4.2.1. Mappa del cluster gerachica
  - 1.4.2.2. Posizionamento di oggetti in RADOS utilizzando CRUSH
  - 1.4.3. Meccanismo di propagazione degli aggiornamenti delle mappe
  - 1.4.4. Replicazione
  - 1.4.5. Consistenza forte
  - 1.4.6. Migrazione dei dati, rilevazione e recupero dai fallimenti
  - 1.4.7. Paxos
  - 1.5. I client RADOS
  - 1.5.1. CephFS
  - 1.5.2. RADOS Gateway
  - 1.5.3. RADOS Block Device
- 2. Setup del cluster ed esempi
  - 2.1. Preparazione macchine
  - 2.2. Bootstrap del cluster
  - 2.3. Aggiunta dei Monitor
  - 2.4. Aggiunta degli OSD
  - 2.5. Verifiche effettuate
  - 2.5.1. Prova del Ceph Storage Cluster tramite la RADOS CLI
  - 2.5.2. Prova di Ceph File System (CephFS)
  - 2.5.3. Prova di Ceph Block Device (RBD)
  - 2.5.4. Prova di Ceph Object Gateway (RadosGW)
  - 2.6. Requisiti minimi in ambiente di produzione
- 3. Stress test
  - 3.1. Disponibilità degli oggetti a seguito di un fallimento di un OSD
  - 3.2. Simulazione blackout
  - 3.3. Scrittura di file di grandi dimensioni tramite CephFS
  - 3.4. Verifica di situazioni di recupero dai fallimenti
- 4. Benchmark: Linda
- 5. Pros & Cons e scenari di utilizzo
- Bibliografia

Introduzione

I sistemi di storage aziendali si sono evoluti enormemente nel tempo; inizialmente si faceva affidamento ad una singola macchina contenente un elevato numero di dischi, in modo da fornire la memoria e le caratteristiche richieste, ma con l'aumenare delle risorse e degli utenti risulta essere una soluzione poco performante. Per risolvere tale problema si è passati all'utilizzo di macchine con hardware molto più potente in termini di prestazioni, ma anche decisamente più costose. Con questa soluzione si è però vincolati ad un solo vendor per tutti i layer dell'architettura di storage, a partire da quello hardware, passando a quello software ed infine per supporto e manutenzione. Per evitare questi svantaggi in termini di costi ed il vendor lock-in, si è pensato ad una soluzione alternativa che invece di utilizzare una sola macchina potente, utilizzasse un cluster di macchine distribuite, ognuna con i propri dischi. In questo modo si possono utilizzare macchine base, comuni, e diminuire drasticamente i costi per la componente hardware; da questa idea nasce Ceph che va a sostituire lo strato software permettendo di astrarre dall'hardware sottostante. In quest'ottica viene tralasciato il layer di supporto e manutenzione che nella soluzione precedente veniva offerto dal vendor, per sostituire questo strato ci si può sottoscrivere ad una qualsiasi azienda che offre questi servizi per Ceph, come ad esempio RedHat. Ceph è una tecnologia open-source che consente la realizzazione di sistemi basati su software-defined storage (SDS). SDS è un approccio alla gestione dei dati nel quale le risorse di storage sono astratte dall'hardware sul quale risiedono; questo rende il sistema più flessibile e programmabile consentendogli di adattarsi automaticamente e rapidamente alle necessità. Ceph è un sistema di storage unificato, completamente distribuito, performante in termini di velocità di trasferimento e latenza, altamente scalabile e senza un "single point of failure"; unificato in quanto consente agli utenti di interfacciarsi coi dati trattandoli come file, blocchi o oggetti astraendo dalla rappresentazione utilizzata dal sistema. La sua architettura è stata pensata basandosi su alcune assunzioni e caratteristiche che si vollero ottenere:

- Ogni componente deve essere scalabile
- Non deve esistere un "single point of failure"
- Open-source, in modo da facilitare la sua espansione
- Concentrato sulla communità, ognuno può decidere quale nuove caratteristiche deve avere, chiunque può risolvere un errore o aggiornare la documentazione
- La soluzione deve basarsi su SDS
- Deve poter essere eseguito su hardware di base
- Qualsiasi cosa deve essere autogestita e autorigenerante quando possibile

Ceph è stato sviluppato all'Università della California da Sage Weil nel 2003 come parte del suo dottorato di ricerca, per poi essere reso open-source nel 2006. Durante l'arco di tempo tra il 2007 e il 2011 Ceph si è raffinato, ottenendo stabilità e affidabilità dei componenti, implementato nuove feature, e stabilendo una roadmap per il futuro. Nel 2012 Weil fondò Inktank per poter permettere la diffusione di Ceph, consentendo alle aziende di implementare e gestire in modo efficace sistemi di storage distribuiti; nel 2014 Inktank è stata acquisita da Red Hat, il più grande fornitore al mondo di soluzioni open-source e fino ad oggi risulta essere uno dei suoi principali curatori.

Un sistema di storage come Ceph è diventato uno dei principali componenti dell'infrastruttura cloud. In particolare risulta essere la tecnologia open-source principale per piattaforme cloud come OpenStack e CloudStack, che sfruttano le sue potenzialità per fornire infrastrutture robuste e scalabili a livello di exabyte. Ceph basa le sue fondamenta sul concetto di oggetto, ogni dato, indipendentemente dal suo formato, viene immagazzinato sotto forma di oggetto all'interno di uno spazio piatto

di uno storage cluster, replicandolo per aumentarne l'affidabilità. Un sistema di storage basato su oggetti ha enormi vantaggi rispetto ai tradizionali sistemi basati su file system; gli oggetti non sono legati ad un percorso fisico e grazie a questa indipendenza dall'hardware sottostante risultano più flessibili e location-independent. Al momento di stesura di questo articolo, sono supportate ancora due versioni di Ceph: Nautilus e Octopus che attualmente sono rispettivamente ai release 14.2.13 e 15.2.5. In seguito faremo sempre e solo riferimento alla versione Octopus release 15.2.5.

1. Architettura e funzionamento di Ceph

1.1. Panoramica generale di Ceph

Nella Figura 1 è possibile vedere una panoramica generale di Ceph. Partendo dal basso vediamo il "commodity hardware" che è inteso come i componenti di qualsiasi macchina base sul quale deve essere installato il sistema operativo Linux. Salendo troviamo l'applicativo Ceph che, installato su ogni macchina, le permetterà di entrare a far parte del CEPH Storage Cluster. Ceph non è altro che una tecnologia che fornisce uno storage cluster basato su oggetti e tre interfacce con le quali l'utente può interagire coi dati: Object Storage, Block Storage e File Storage.

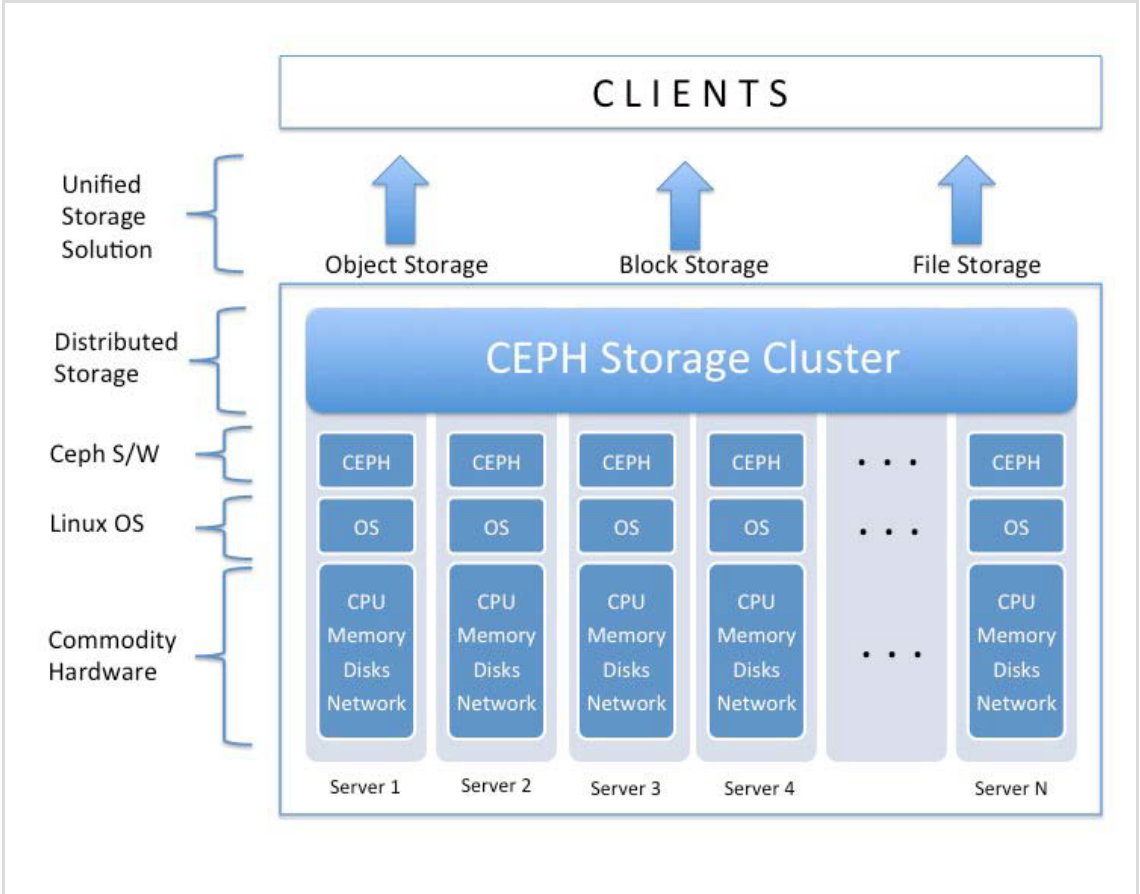

Figura 1 - Panoramica generale di Ceph

1.2. Panoramica tecnica di Ceph

Entrando più nel dettaglio è possibile dare una panoramica un po' più tecnica (vedi Figura 2) assegnando dei nomi ai componenti visti nell'immagine precedente. Alle fondamenta di Ceph troviamo RADOS (Reliable Autonomic Distributed Object Store) che è lo storage cluster di Ceph, cioè il componente sul quale si basa tutto il suo funzionamento; in questo sono implementate tutte le funzionalità sui dati, come la gestione degli snapshots, la replicazione ed il provisioning, e tutta la loro logica di memorizzazione e distribuzione. RADOS utilizza un algoritmo chiamato CRUSH (Controlled Replication Under Scalable Hashing) per determinare come mappare e replicare i dati all'interno dei singoli nodi. Al di sopra di RADOS si possono inoltre vedere le interfacce con cui gli utenti potranno interagire, prendendo come riferimento la Figura 1 si ha RadosGW per l'object storage, RBD per il block storage ed infine CephFS per il file storage; in aggiunta a questi vi è anche Librados che nella Figura 1 non era mostrata. Librados è una libreria che può essere utilizzata per accedere direttamente a RADOS e alle sue funzionalità, non è una interfaccia come le precedenti ma è un protocollo nativo, e di conseguenza è molto più veloce; inoltre RadosGW e RBD utilizzano questa libreria per interagire con RADOS. Nel caso un utente voglia interagire con lo storage cluster utilizzando gli oggetti potrà sfruttare l'interfaccia RadosGW o la libreria Librados, mentre se volesse operare con blocchi di memoria avrebbe a disposizione RBD ed infine per memorizzare file o cartelle organizzati in una struttura ad albero ci sono le funzionalità fornite da CephFS.

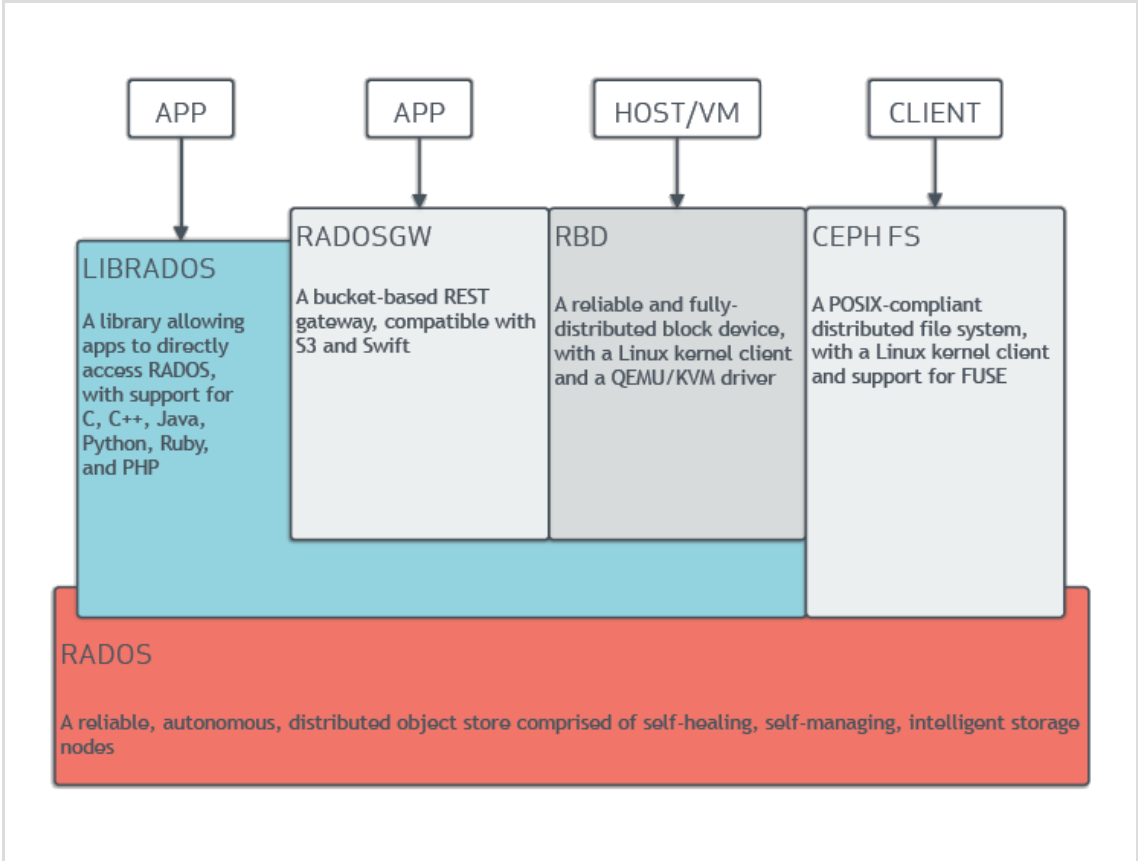

Figura 2 - Panoramica tecnica di Ceph

1.3. Componenti dello storage cluster Ceph

I componenti fondamentali, indispensabili per il funzionamento di Ceph, sono due: Monitor e OSD (Object Storage Daemon); a questi si aggiungono altri componenti che

vengono utilizzati per accedere a tutte le funzionalità dello storage cluster. Tra quelli in rilievo troviamo i Manager, che vengono utilizzati per il tracciamento delle metriche e dello stato del cluster, e gli MDS (Metadata Server), che sono indispensabili per accedere agli oggetti tramite i client Ceph File System.

1.3.1. Manager

I Manager sono componenti responsabili di mantenere traccia delle informazioni riguardanti placement groups, host, metadata dei processi e delle metriche relative l'esecuzione e lo stato corrente del cluster Ceph, come ad esempio l'utilizzo della memoria, performance ed il carico del sistema. Questi componenti vengono affiancati ai monitor per aiutarli nella gestione e nel monitoraggio del sistema; contengono moduli Python per gestire ed esporre informazioni sul cluster, incluse una Dashboard web e delle API REST. Per ottenere le funzionalità di questo componente è necessario solamente un Manager, ma per ottenere un sistema ad alta disponibilità ne sono richiesti almeno due; in questo secondo caso il fallimento di un manager non pregiudicherà le funzionalità offerte al sistema.

1.3.2. Monitor

I Monitor sono componenti che si occupano di monitorare lo stato degli OSD e di gestire tutte le mappe relative al cluster; queste mappe vengono utilizzate per supportare il coordinamento ed il funzionamento dei servizi Ceph, inoltre ne verrà richiesta una copia dai client per interagire con lo storage cluster. Le mappe mantenute dai monitor, che assieme compongono la "cluster map", sono le seguenti:

- mappa degli OSD, che contiene l'fsid del cluster, quando è stata creata la mappa, quando è stata fatta l'ultima modifica, la lista dei pool, la dimensione delle repliche, il numero dei PG e una lista degli OSD ed il loro stato;
- mappa dei monitor, che contiene l'fsid del cluster assieme alla posizione, nome, indirizzo e porta di ogni monitor. Inoltre indica l'epoca corrente, quando la mappa è stata creata e l'ultima volta in cui è stata modificata;
- mappa degli MDS, che contiene l'epoca corrente, quando è stata creata, l'ultima volta che è stata modificata, il pool per memorizzare i metadati, una lista dei server di metadati ed il loro stato;
- mappa dei Placement Group, che contiene la versione dei PG, il suo timestamp, l'ultima epoca della mappa degli OSD, le "full ratios" e i dettagli su ogni placement group come il suo id, il suo Up Set, l'Acting Set, lo stato del PG (ad esempio active+clean) e statistiche sull'uso dei dati di ciascun pool;
- mappa CRUSH, che contiene una lista dei dispositivi di memorizzazione, il livello di fallimento (ad esempio dispositivo, host, rack, ecc) e le regole per memorizzare i dati.

Per il corretto funzionamento deve essere garantita alta consistenza tra le mappe di tutti i monitor, per far ciò viene utilizzato Paxos come algoritmo di consenso, in modo tale da assicurarsi che siano tutti in accordo sullo stato attuale del cluster. I monitor vengono utilizzato per il consenso quando devono essere prese decisioni distribuite, fornendo così una regola generale per tali decisioni particolari riguardanti gli stati degli OSD; sono inoltre responsabili della gestione dell'autenticazione, sfruttando il protocollo di autenticazione cephx permettono a client e ai servizi di autenticarsi ed accedere alle funzionalità del cluster. Le impostazioni del cluster di archiviazione Ceph possono essere gestite principalmente in due modi: attraverso file di configurazione locali o attraverso un database centralizzato gestito dai monitor. È consigliato utilizzare il database condiviso per la maggior parte delle impostazioni e mantenere nei file di configurazione locali

solamente ciò che serve per permettere agli altri componenti di connettersi, autenticarsi e recuperare le informazioni di configurazione dal database. Il file di configurazione locale di default è situato all'interno di /etc/ceph/ ed è nominato ceph.conf. Avere un database centralizzato gestito dai monitor aiuta a migliorare la gestione delle configurazioni dell'intero cluster di archiviazione Ceph e porta molti aspetti positivi come ad esempio il fatto di non dover propagare manualmente una configurazione in caso di modifica. Per il funzionamento di un cluster è necessario solamente un monitor, ma per ottenere ridondanza ed alta disponibilità ne sono richiesti almeno tre. É consigliato l'utilizzo di un numero dispari e non troppo elevato di questi componenti per agevolare il meccanismo di raggiungimento del quorum in quanto: con un numero dispari è garantito che venga trovata la maggioranza ed un elevato numero di monitor è controproducente in quanto rallenterebbe il processo di raggiungimento del consenso. Inoltre con 3 monitor può essere tollerato il fallimento di uno di essi in quanto il meccanismo necessita di un numero di monitor attivi superiore alla metà per poter funzionare; ad esempio, con 2 monitor non sono tollerati fallimenti, con 3 è tollerato un solo fallimento, con 4 è tollerato un solo fallimento, con 5 sono tollerati 2 fallimenti e così via.

1.3.3. Metadata Server

I Server Metadata (MDS) sono componenti che hanno il compito di memorizzare i metadati utilizzati dal File System Ceph (CephFS), di conseguenza sono necessari solamente nel caso in cui si voglia accedere al cluster di archiviazione attraverso CephFS (non sono utilizzati con Block Devices ed Object Storage). Questi componenti sono utilizzati per permettere agli utenti di accedere a tutte le funzionalità del file system POSIX (come ad esempio i comandi ls, find, ecc..) senza appesantire eccessivamente il cluster di archiviazione Ceph.

1.3.4. Object Storage Daemon

Gli Object Storage Daemon (OSD) sono componenti che si occupano di immagazzinare i dati e gestire operazioni su di essi, come la replicazione ed il recupero dei dati, il ribilanciamento e backfilling e l'erasure coding. Inoltre si occupano anche di fornire informazioni di monitoraggio ai servizi monitor e manager, richiedendo agli altri OSD presenti nel cluster un segnale di heartbeat per controllare il loro status ed identificare eventuali anomalie. Ogni nodo del cluster esegue uno o più OSD, ciascuno associato ad un disco fisico o virtuale; l'insieme di tutti gli OSD forma lo Storage Cluster. Per il corretto funzionamento di uno storage cluster sono richiesti almeno 3 OSD in modo da garantire ridondanza e alta disponibilità di servizio.

1.4. RADOS ed il suo funzionamento

RADOS (Reliable Autonomic Distributed Object Store), come afferma il suo acronimo, è uno storage distribuito di oggetti altamente scalabile che si pone come obiettivo quello di essere automatico e affidabile; esso è composto da nodi intelligenti, autorigeneranti ed autogestiti. In esso è contenuta la business logic di Ceph, cioè tutta la logica di memorizzazione, distribuzione e gestione dei dati; inoltre offre alcune funzionalità come la gestione della replicazione e dell'erasure coding (per tutelare il sistema da perdite di dati a seguito del fallimento di un nodo), il ribilanciamento in caso di fallimento di un host ed altre. RADOS permette di facilitare il bilanciamento della distribuzione dei dati e del carico di lavoro su un cluster di macchine dinamico ed eterogeneo fornendo alle applicazioni l'illusione di operare con un singolo storage logico di oggetti, con una semantica ben definita e garantendogli consistenza. Viene fornita una mappa versionata del cluster a tutti i

nodi, che siano client o di storage, garantendo così una vista consistente del cluster e della distribuzione dei dati; dato che questa mappa potrebbe cambiare frequentemente, viene replicata su tutti i nodi (proprietà di consapevolezza dell'infrastruttura) e verrà aggiornata propagando dei piccoli aggiornamenti incrementali per mantenerla consistente. La conoscenza completa della distribuzione dei dati incapsulata dalla cluster map permette a RADOS di distribuire, tra gli OSD che compongono lo storage cluster, la gestione della replicazione dei dati, il loro aggiornamento in modo consistente e sicuro e la rilevazione ed il recupero dei fallimenti; permettendo quindi a questi di agire in modo peer-to-peer semi-automatico. La mappa del cluster include una descrizione e lo stato corrente dei dispositivi su cui sono distribuiti i dati; questo include l'indirizzo di rete attuale di tutti gli OSD che sono attualmente online e raggiungibili, detti *up*, e di quelli che invece non lo sono, detti *down*. RADOS considera un'ulteriore dimensione, la liveness degli OSD: i device *in* sono inclusi nel mapping e ad essi possono essere assegnati placement group, mentre i dispositivi *out* non verranno considerati per la distribuzione dei dati. Per ogni PG, CRUSH produce una lista di OSD che fanno parte del cluster, da cui RADOS poi rimuoverà i dispositivi che sono attualmente *down* per produrre la lista degli OSD attivi per il PG dato. Se la lista è vuota allora i dati del PG sono temporaneamente non disponibili e le richieste di I/O che sono in attesa verranno bloccate.

I dati sono memorizzati all'interno di RADOS sotto forma di uno o più oggetti (in base alle dimensioni) organizzati in uno spazio piatto; ogni oggetto tipicamente è formato da 3 parti: un identificativo (univoco all'interno del cluster), i dati da memorizzare ed i metadati (ad esempio le informazioni riguardo la sua creazione). Gli oggetti sono memorizzati in pool; ogni pool è definito da un nome ed identifica uno spazio dei nomi degli oggetti distinto, questo significa che potranno essere salvati oggetti con lo stesso nome in pool diversi mentre nello stesso pool non possono esistere omonimi. Ogni pool inoltre contiene alcuni parametri che definiscono come gli oggetti vengono memorizzati, come il numero di repliche che dovranno essere realizzate e come queste dovranno essere distribuite nel cluster.

Un sistema RADOS è composto da un numero più o meno grande di OSD (dal quale dipenderà la capacità dello storage cluster) ed un piccolo gruppo di monitor. Gli OSD vengono utilizzati per gestire i dischi che fanno parte dello storage cluster e sono gli unici componenti del cluster Ceph che memorizzano gli oggetti degli utenti; a ogni disco che deve essere aggiunto al cluster (che esso sia un hard disk, un SSD, un gruppo RAID o altro) viene assegnato un OSD che ne gestisce lo stato ed effettua tutte le operazioni sui dati, come ad esempio scritture, letture, replicazione e recupero. Una caratteristica chiave della progettazione di RADOS è che gli OSD sono in grado di funzionare con una relativa autonomia quando si tratta di recuperare da errori o migrare dati in risposta all'espansione o restringimento del cluster. Per cui il sistema risulta estremamente scalabile, può crescere senza problemi fino a centinaia o migliaia di nodi, secondo necessità, o restringersi nel caso sia sovradimensionato. I monitor hanno il compito di tracciare lo stato dell'intero cluster e di mantenere la Cluster map, che è definita come l'insieme di tutte le mappe relative al cluster. La mappa specifica quali OSD fanno parte del cluster, il loro stato e come sono distribuiti i dati su questi dispositivi; inoltre ogni singola mappa che la compone contiene un valore epoca che viene incrementato ogni volta che queste subiscono una modifica, a seguito di cambio di stato di OSD o di qualsiasi evento che alteri il layout dei dati. Questo valore permette a tutti coloro che necessitano di interagire con il cluster di determinare se la loro replica è aggiornata o meno, e di concordare su quale sia la distrubuzione dei dati corrente; se la loro replica non fosse aggiornata sarà necessario allinearsi attraverso dei messaggi di aggiornamento, nel

caso in cui la distanza tra le due mappe sia di più epoche gli aggiornamenti possono essere raggruppati per descrivere le differenze tra queste.

1.4.1. Posizionamento dei dati

Un aspetto molto importante e che distingue Ceph dai sistemi tradizionali di storage è relativo alla metodologia di posizionamento dei dati. Nei sistemi tradizionali i client interagiscono con un componenete centralizzato per scrivere o leggere dati, solitamente chiamato gateway o controller; questo componente, dato che è l'unico punto di ingresso nel sistema, risulta essere un single point of failure in quanto il suo fallimento si ripercuoterebbe sull'intero sistema. Questa architettura pone inoltre limiti a livello di scalabilità e performance. A differenza di tali sistemi, Ceph permette ai client di interagire direttamente con gli OSD eliminando il componente centralizzato e di conseguenza aumentando la sicurezza e la disponibilità; in quest'ottica i client dovranno però conoscere quali sono gli OSD con i quali interagire e per farlo potranno ottenere informazioni su come è strutturato il cluster da un gruppo di monitor.

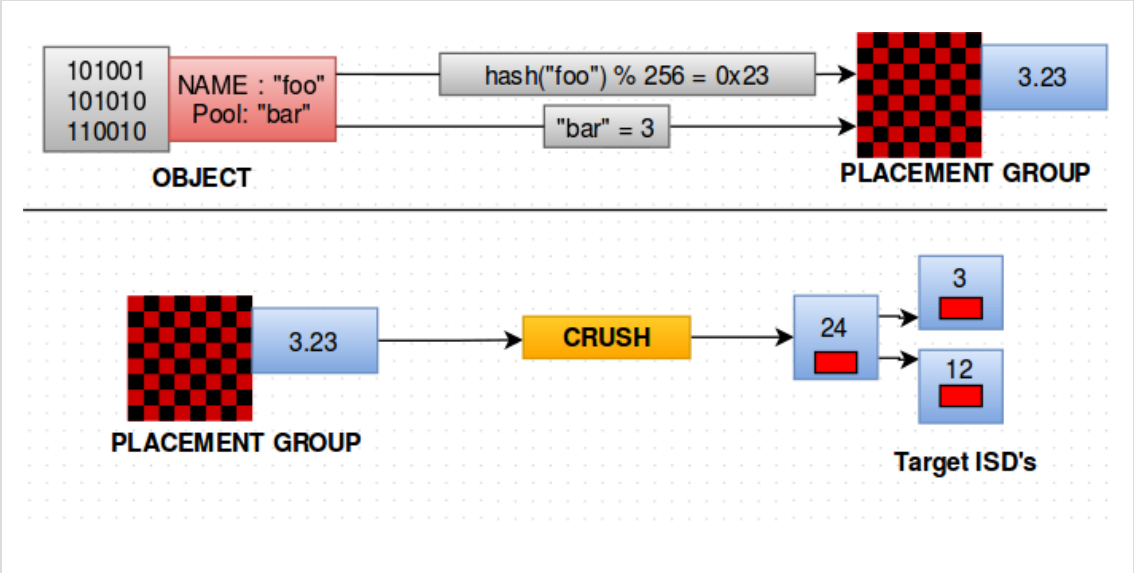

Figura 3 - Posizionamento dei dati

Invece di memorizzare e manipolare metadati, RADOS utilizza un nuovo metodo per determinare il posizionamento dei dati: l'algoritmo CRUSH (Controlled Replication Under Scalable Hashing); tramite questo algoritmo è possibile individuare con quale OSD dovrà interagire un client per leggere o scrivere un oggetto. Il calcolo della posizione degli oggetti (vedi Figura 3) viene fatto direttamente dal client conoscendo il nome dell'oggetto, il nome del pool, lo stato del cluster e l'insieme delle regole CRUSH; mentre i primi due sono già conosciuti dal client, gli ultimi due devono essere ottenuti interagendo con il cluster dei monitor. Questa è un'operazione altamente veloce e performante dato che viene effettuata direttamente dal client e non necessita della presenza di un controller. Prima di proseguire è necessario capire alcuni concetti come quello di pool e placement group (panoramica grafica in Figura 4); un pool è una partizione logica dell'intero cluster utilizzata dai client per memorizzare oggetti, mentre i Placement Group (PG) sono sottoinsiemi logici di oggetti completamente invisibili al client, ma che giocano un ruolo fondamentale nel funzionamento dello storage cluster. Lo storage cluster deve essere in grado di crescere (o diminuire) e ribilanciarsi memorizzando oggetti dinamicamente, e dato che

il numero degli oggetti memorizzati in un pool può facilmente superare i milioni, tracciare il posizionamento degli oggetti in un'ottica "per oggetto" diventa un'operazione computazionalmente onerosa da scalare; inoltre senza un livello di astrazione si verrebbe a creare un accoppiamento stretto tra il client e l'OSD dato che il client dovrebbe conoscere in quale OSD è situato ogni oggetto. Per aumentare le performance allo scalare del sistema, Ceph suddivide i pool in placement group, assegnando ogni oggetto ad un placement group e mappando ogni PG ad un OSD primario; questo crea un livello di indirezione tra gli OSD ed i client, consentendo a Ceph di ribilanciarsi dinamicamente quando si aggiunge un nuovo OSD o uno di questi fallisce. In questo modo il cluster sarà in grado di ribilanciarsi in modo dinamico ed efficiente, a seguito di un fallimento o dell'aggiunta di un OSD, muovendo o replicando interi placement group invece di dover reindirizzare ogni oggetto individualmente.

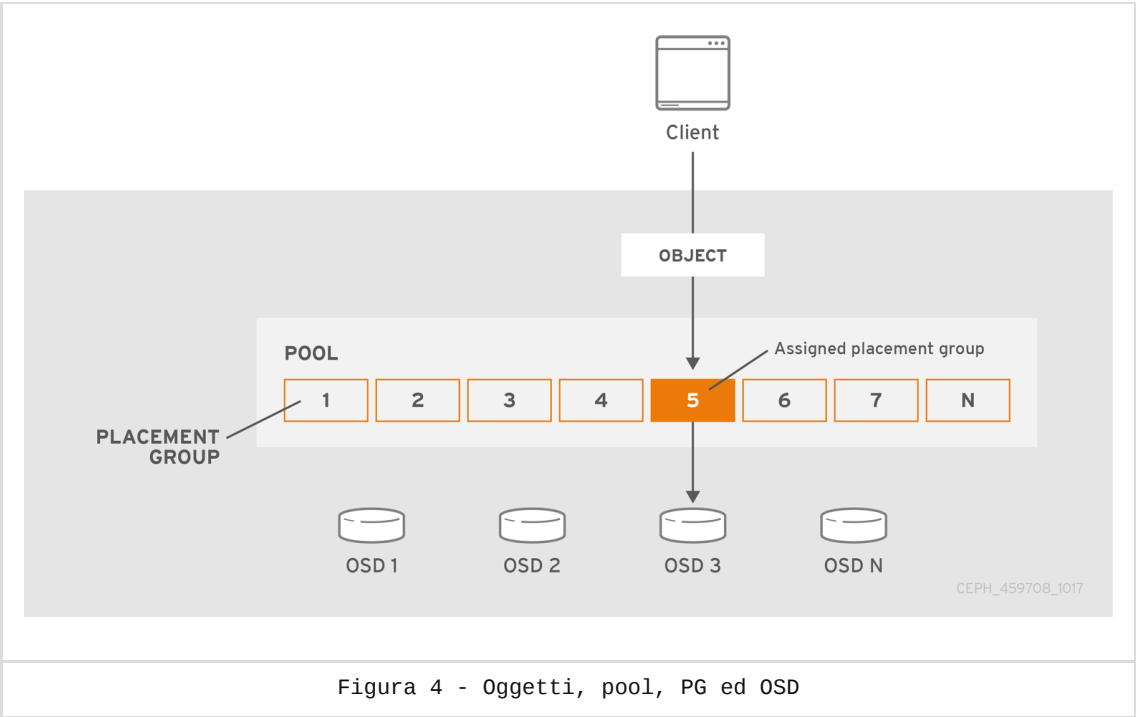

Figura 4 - Oggetti, pool, PG ed OSD

È possibile dividere il meccanismo di posizionamento dei dati in due fasi:

- con il nome dell'oggetto e del pool viene calcolato l'ID del placement group dell'oggetto;
- con l'ID del PG, l'insieme delle regole CRUSH e lo stato del cluster viene utilizzato l'algoritmo CRUSH per ottenere una lista di OSD, dove il primo viene definito primario ed è quello con il quale il client interagirà.

Una volta richiesta la scrittura all'OSD primario, questo effettuerà tale operazione sul proprio supporto di memorizzazione e richiederà la replica del PG agli altri OSD della lista, non appena ricevuta la conferma dell'avvenuta scrittura dagli altri OSD, questo confermerà al client il successo dell'operazione. Questo sposta la responsabilità di aggiornare tutte le repliche consistentemente ed in modo sicuro sugli OSD ed ha il vantaggio che effettuando tale operazione all'interno del cluster si semplifica la progettazione del client e si sposta la banda necessaria all'interno della rete dello storage cluster.

1.4.2. CRUSH (Controlled Replication Under Scalable Hashing)

Nella maggior parte dei sistemi i nuovi dati vengono semplicemente scritti sui dispositivi meno utilizzati. Questo approccio presenta un problema fondamentale, in quanto i dati una volta scritti raramente vengono spostati, per cui anche una distribuzione inizialmente perfetta diventerà, prima o poi, sbilanciata quando il sistema sarà inevitabilmente espanso, dato che i nuovi dischi saranno vuoti o conterranno solamente nuovi dati. Una soluzione robusta a questo problema è quella di distribuire tutti i dati casualmente tra tutti i device disponibili, portando così ad una distribuzione probabilistica bilanciata che mischia nuovi e vecchi dati. Questo ha come vantaggio di avere tutti i dispositivi con un carico simile, permettendo al sistema di performare bene sotto un qualsiasi carico di lavoro.

Per ottenere questa situazione Ceph fa uso dell'algoritmo CRUSH, che è implementato come una funzione pseudo-random, deterministica e ripetibile, che mappa un valore di input (tipicamente un identificativo di un oggetto o gruppo di oggetti) ad una lista di dispositivi su cui immagazzinare l'oggetto e le sue repliche. È pensato per distribuire i dati ottimalmente sulle risorse disponibili, riorganizzazdoli nel caso di aggiunta o rimozione di dispositivi di storage, facendo valere vincoli flessibili sul piazzamento delle repliche per massimizzare la sicurezza dei dati nel caso di malfunzionamenti dell'hardware.

In aggiunta a tutto ciò, CRUSH presenta la caratteristica di consapevolezza dell'infrastruttura sottostante, in quanto conosce e comprende le relazioni tra i vari componenti; esso immagazzina i dati in modo tale che siano sempre disponibili, anche in caso di malfunzionamento di alcuni elementi del sistema, garantendo così affidabilità e robustezza. L'algoritmo inoltre permette a Ceph di essere auto-gestito e auto-riparante nell'eventualità di un malfunzionamento, permettendogli di capire quale componente ha generato il problema ed effettuare operazioni di recupero sui dati, replicando le copie dei PG presenti nell'OSD fallito in altri OSD disponibili.

1.4.2.1. Mappa del cluster gerachica

Ad ogni dispositivo di storage viene assegnato un peso dal quale dipende la distribuzione degli oggetti effettuata tramite l'algoritmo CRUSH, questo consente di approssimare una distribuzione probabilistica uniforme. Tale distribuzione è controllata da una mappa del cluster gerarchica che rappresenta le unità di storage disponibili. La politica di distribuzione dei dati è definita in termini di regole di posizionamento che specificano quante repliche dovranno essere effettuate e quali restrizioni sono imposte al loro posizionamento. Ad esempio si potrebbe specificare che tre repliche debbano essere piazzate in device che risiedono in tre luoghi fisici differenti, in modo da mitigare problemi come ad esempio blackout localizzati.

La cluster map è composta da dispositivi e bucket, entrambi con identificatori numerici e pesi associati. I bucket possono contenere un qualsiasi numero di dispositivi o anche altri bucket, permettendo così la creazione di nodi interni alla gerarchia di storage, in cui i dispositivi sono sempre situati ai nodi foglia. L'amministratore del sistema può assegnare ai dispositivi dei pesi che rappresentano e controllano l'ammontare di dati che essi avranno il compito di immagazzinare (normalmente derivati dalle capacità del dispositivo); i pesi dei bucket sono definiti dalla somma di tutti i pesi dei dispositivi e bucket che ne fanno parte.

I bucket possono essere definiti arbitrarialmente per costruire una gerarchia che rappresenti lo storage fisico disponibile. Ad esempio, si potrebbe creare una mappa contenente dei bucket chiamati "scaffale" al livello più basso, che rappresentano insiemi di device, e poi combinarli in altri bucket "armadietto" per raggruppare assieme gli scaffali che sono installati nello stesso rack.

Number	CRUSH Bucket	Description
0	**OSD**	An OSD daemon (for example, osd.1, osd.2, and so on).
1	**Host**	A host name containing one or more OSDs.
2	**Rack**	A computer rack containing one or more hosts.
3	**Row**	A row in a series of racks.
4	**Room**	A room containing racks and rows of hosts.
5	**Data Center**	A physical data center containing rooms.
6	**Root**	This is the beginning of the bucket hierarchy.

Figura 5 - Tipologie di bucket base di Ceph

I dati sono poi inseriti all'interno della gerarchia, selezionando ricorsivamente i bucket innestati, attraverso funzioni hash-like e pseudo-random, che differiscono tra loro in termini di performance ed efficienza riorganizzativa, in particolare queste sono:

- Uniform: può essere usato se tutti i device di storage hanno lo stesso peso, ad esempio per hardware che viene commissionato o decommissionato in gruppo. Questo algoritmo permette a CRUSH di mappare repliche all'interno di bucket uniformi in tempo costante. Non è assolutamente conveniente da usare in caso di pesi non uniformi, inoltre l'aggiunta o rimozione di dispositivi all'interno di questo tipo di bucket è da evitare in quanto causerebbe un totale rimescolamento dei dati, rendendolo poco ideale per cluster le cui dimensioni possono variare nel tempo.
- List: i bucket aggregano il proprio contenuto all'interno di liste linkate. Questa scelta è ottima per cluster in espansione, in quanto l'aggiunta di un nuovo device provocherebbe un ottimale movimento dei dati. Questo però non accade nel caso di rimozione di device dalla coda o all'interno della lista, rendendo l'algoritmo ideale per cluster che si riducono raramente in dimensioni, o ancora meglio mai.
- Tree: questi tipi di bucket utilizzano un albero di ricerca binario, e sono più efficienti del tipo List per quanto riguarda ampi insiemi di oggetti. I bucket di questa tipologia sono strutturati come alberi di ricerca binari pesati, in cui ogni nodo centrale conosce i pesi dei sottoalberi alla sua destra e sinistra, ed è etichettato a seconda di una strategia prestabilita.
- Straw: i bucket di questo tipo permettono a tutti gli oggetti di "competere" fra di loro equamente per il piazzamento delle repliche, consentendo un comportamento ottimale di migrazione tra i sottoalberi. Il processo di competizione tra gli oggetti è analogo al processo di estrazione delle pagliuzze. Per piazzare una replica, viene estratta per ogni oggetto del bucket una pagliuzza di lunghezza casuale, e sarà selezionato l'oggetto che estrarrà la più lunga. Anche se questo processo è più lento rispetto gli altri algoritmi, circa il doppio in confronto a list e ancora di più considerando

tree (dato che scala logaritmicamente), i bucket di tipo straw risultano ottimali per il movimento dei dati tra oggetti innestati. Ceph utilizza l'algoritmo straw di default per i suoi bucket.

La scelta del tipo di bucket deve essere definita sulla base del fattore di crescita del cluster. Quando ci si aspetta di avere bucket di dimensione fissa, il tipo uniform è il più veloce, mentre se si pensa che il bucket possa soltanto crescere in dimensioni allora il tipo list permette il movimento di dati ottimale, permettendo a CRUSH di spostare l'ammontare appropriato di dati tra i dispositivi, senza dover rimescolare dati da altri bucket. Nella situazione nella quale ci si aspetta che l'operazione più frequente sia la rimozione di dispositivi e che l'efficienza nella riorganizzazione sia critica, risulta più adatto l'algoritmo straw, che presenta un comportamento ottimale per le migrazioni fra sottoalberi. Infine il tipo tree offre un ottimo compromesso, fornendo performance elevate e buone capacità riorganizzative. Nella figura seguente si può osservare un confronto in termini di velocità delle operazioni e costo dell'aggiunta e rimozione di dispositivi tra le varie tipologie di algoritmo appena descritte.

Action	Uniform	List	Tree	Straw
Speed	O(1)	O(n)	O(log n)	O(n)
Additions	poor	optimal	good	optimal
Removals	poor	poor	good	optimal

Figura 6 - Complessità algoritmi di selezione bucket

1.4.2.2. Posizionamento di oggetti in RADOS utilizzando CRUSH

Entrando più nel dettaglio possiamo esaminare come RADOS utilizza CRUSH per determinare il posizionamento degli oggetti all'interno del cluster. Ogni PG di un oggetto è determinato dal modulo tra l'hash del nome dell'oggetto ed il numero dei PG, appeso poi al pool ID per trovare l'ID del placement group che ospiterà l'oggetto; questo può essere fatto dato che un oggetto viene sempre immagazzinato in un pool concreto, quindi l'ID del pool ed il nome dell'oggetto identificano univocamente un oggetto all'interno del sistema. Con l'ID del PG, lo stato del cluster e le ruleset CRUSH (o crushmap) è possibile utilizzare l'algoritmo CRUSH per ottenere una lista di OSD dove la sua lunghezza corrisponderà al numero totale di copie che si vogliono ottenere (copia originale + copie replica). Una volta che l'oggetto è stato scritto sull'OSD primario, questo repricherà il PG sugli altri OSD della lista ed una volta ricevuti gli ack da questi confermerà l'avvenuta scrittura al client.

Quando un pool viene creato, gli viene assegnato un numero di placement group che potrà variare nel corso della sua vita in base alla capacità che gli verrà attribuita. Ad esempio i passi effettuati per recuperare la lista degli OSD per memorizzare un oggetto di nome "tree" nel pool "images", che contiene 65536 placement groups sono i seguenti:

1. Effettuare l'hash del nome dell'oggetto, ad esempio hash('tree')= 0xA062B8CF
2. Calcolare il modulo tra l'hash ed il numero di PG: 0xA062B8CF % 65536 = 0xB8CF
3. Ottenere l'ID dal nome del pool: 'images' = 7
4. Prefiggere l'ID del pool al valore calcolato nel secondo passo per ottenere l'ID del placement group: 7.B8CF

5. Ceph infine utilizzerà l'ID del placement group, assieme alla mappa del cluster e le regole di posizionamento per ottenere la lista degli OSD: CRUSH('7.B8CF') = [4,19,3].

La dimensione di questa lista è data dal numero di copie che si vogliono ottenere ed è configurato per il pool in questione, questo numero comprende la copia originale ed il numero di repliche che si vogliono ottenere. Tramite questa lista Ceph è in grado di ricomporre un oggetto; il primo OSD della lista è detto primario ed è l'unico che risponderà direttamente alle richieste dei client.

1.4.3. Meccanismo di propagazione degli aggiornamenti delle mappe

Come precedentemente anticipato, le mappe vengono replicate nei nodi client e di storage quando questi interagiranno col cluster per la prima volta, per poi essere aggiornate con piccoli aggiornamenti incrementali. Poiché lo storage cluster può essere composto da molte migliaia di dispositivi potrebbe non essere pratico trasmettere gli aggiornamenti delle mappe in modalità broadcast a tutti; per questo motivo si è pensato ad una distribuzione "lazy" degli aggiornamenti. Dato che le differenze nelle epoche delle mappe sono significative solo quando variano tra due OSD comunicanti è possibile spostare il carico di distribuzione degli aggiornamenti delle mappe sugli OSD, combinandoli con i messaggi già scambiati tra di essi. Per far ciò ogni OSD mantiene traccia dell'epoca più recente che ha osservato fino a quel momento e di uno storico degli aggiornamenti passati delle mappe, e contrassegna tutti i messaggi con la sua ultima epoca. Se un OSD riceve un messaggio con una mappa passata, risponde a quel peer, cioè un'altro OSD con cui condivide le informazioni dei PG, condividendogli gli incrementi necessari per sincronizzarlo. Inoltre, se un OSD contatta un peer che si ritiene abbia un'epoca precedente, gli condivide preventivamente gli aggiornamenti incrementali; questa strategia viene definita conservativa in quanto gli OSD riceveranno aggiornamenti duplicati da tutti i dispositivi che non sono certi che questo sia sincronizzato. Il livello effettivo di duplicazione degli aggiornamenti è molto inferiore al numero di PG che gestisce, dal quale dipende il numero di peer che possiede. I messaggi heartbeat scambiati periodicamente per il rilevamento degli errori assicurano che gli aggiornamenti si diffondano rapidamente, in un tempo O(log(n)) per un cluster di n OSD. Nel caso un OSD che fa parte del cluster torni *up*, informerà un monitor del fatto che è tornato disponibile con un messaggio "OSDBoot" che include la sua epoca più recente. Il cluster di monitor cambierà lo stato dell'OSD e risponderà con gli aggiornamenti incrementali necessari a sincronizzare completamente l'OSD. Quando il nuovo OSD inizierà a contattare gli altri per condividere dati, propagherà anche gli ultimi aggiornamenti che possiede, in modo tale da aggiornare tali dispositivi; dato che, essendo appena tornato *up*, non conoscerà esattamente quali epoche hanno gli altri OSD, condividerà uno storico composto dagli ultimi aggiornamenti delle mappe.

1.4.4. Replicazione

RADOS è stato progettato per poter utilizzare 3 diverse strategie di replicazione: primary-copy, chain, e splay; tutte e tre le strategie forniscono forti garanzie di coerenza, in modo tale che le operazioni di lettura e scrittura avvengano in un certo ordine sequenziale e che le scritture completate si riflettano nelle successive letture. In Figura 7 è possibile vedere i messaggi scambiati durante un'operazione di aggiornamento distinguendo le diverse strategie; in tutti i casi i client inviano le operazioni di scrittura all'OSD primario (il primo nella lista degli OSD ritornata da CRUSH) ed il cluster garantisce che le repliche siano aggiornate in modo sicuro e che sia conservata la consistenza della semantica di lettura/scrittura. Una volta che

tutte le repliche sono state aggiornate verrà spedito un singolo messaggio ack al client per confermare il successo dell'operazione richiesta. La replicazione primary-copy aggiorna tutte le repliche in parallelo e prevede che i client inviino sia le letture che le scritture all'OSD primario. Le scritture saranno reindirizzate in parallelo agli altri dispositivi in cui dovranno essere replicate, dove questi applicheranno gli aggiornamenti nel loro storage locale e risponderanno al primo OSD. Una volta ricevute le risposte che tutte le repliche sono state salvate, l'OSD primario applica l'aggiornamento nel suo storage locale e risponde al client. La replicazione chain aggiorna le repliche in serie: le scritture vengono inviate all'OSD primario (testa) che applica l'aggiornamento localmente e lo inoltra al successivo OSD nella lista. L'ultimo OSD (la coda), dopo aver aggiornato il proprio storage locale, risponde al client del'avvenuta operazione. Le letture invece vengono inviate alla coda, nella quale le risposte rispecchieranno sempre l'ultimo aggiornamento interamente replicato. Rispetto la strategia precedente ha il vantaggio di necessitare di meno salti di rete e messaggi scambiati (3 rispetto ai 4 del primary-copy) ma ha lo svantaggio che la latenza aumenta proporzionalmente all'allungarsi della catena (mentre nella precedente le replicazioni vengono fatte in parallelo quindi rimane costante) rendendo questa strategia problematica per alti livelli di replicazione. La replicazione splay è un ibrido tra le strategie precedenti e combina gli aggiornamenti paralleli del primary-copy con la separazione dei ruoli di lettura/scrittura del chain. Come per chain le scritture saranno dirette al primo OSD mentre le letture all'ultimo; a differenza del primary-copy, il primario esegue l'aggiornamento nel proprio storage locale appena questo gli viene comunicato e in seguito lo propaga in parallelo agli altri OSD. A questo punto, gli OSD che riceveranno gli aggiornamenti effettueranno la memorizzazione locale e lo comunicheranno all'ultimo OSD (la coda), che, una volta ricevute tutte le risposte, aggiornerà il proprio storage e notificherà al client la terminazione dell'operazione richiesta. Sia primary-copy che splay ritardano la propria scrittura locale fino alla ricezione delle risposte dagli altri dispositivi per mantenere alta consistenza in caso di fallimento di un OSD, ma a differenza del primo, splay necessita di meno tempo riducendo la memoria richiesta per tale operazione.

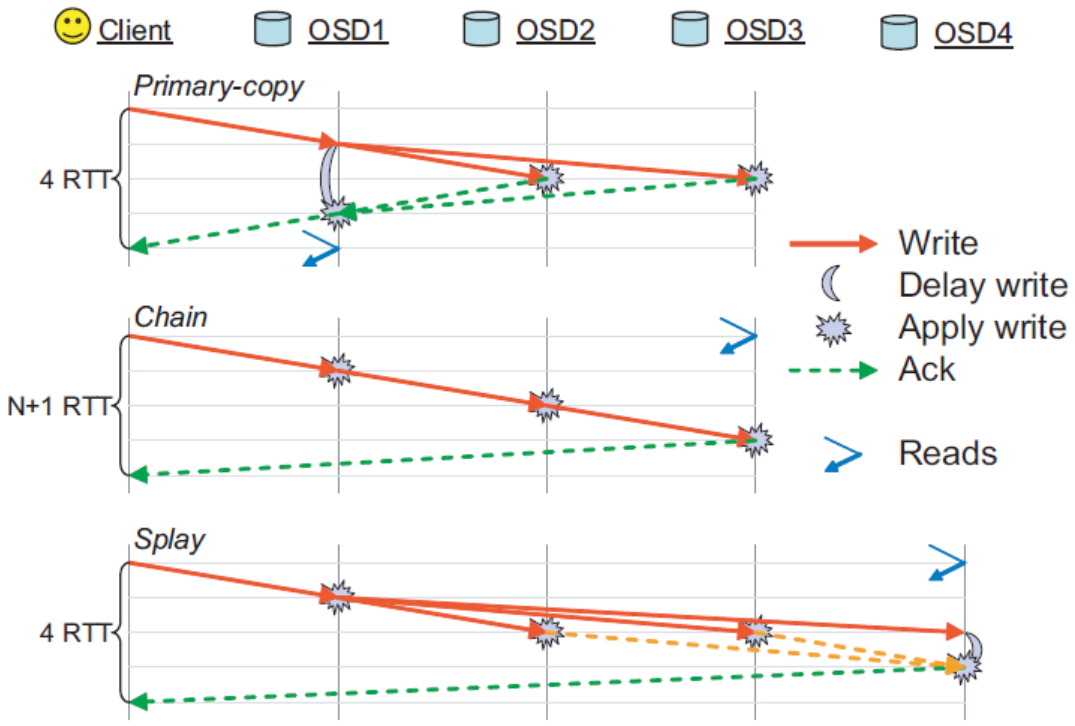

Figura 7 - Strategie di replicazione di RADOS

Erasure coding

Eresure coding è una tecnica per garantire la proprietà di fault tollerance, questa prevede di prendere i dati originali, dividerli in chunk e codificarli in un qualche modo per far sì che nel caso uno o più dei chunk venisse a mancare si potrebbero usare questi pezzi codificati al posto di quelli mancanti per ricostruire i dati iniziali. Si può pensare all'erasure coding come un sistema di più equazioni dove le parti originali sono le incognite e le equazioni i chunk; ad esempio avendo il dato iniziale 64 dove x=6 e y=4 potremmo pensare di aggiungere una o più equazioni al sistema per ricondurci ai dati originali in caso di perdita delle informazioni iniziali, come x-y=2 o x+y=10. Aggiungendo uno delle due equazioni al sistema otterremmo un sistema a tre equazioni e due incognite quindi per ricomporre il dato originale avremmo bisogno solo di due dei tre pezzi. In Ceph viene utilizzata questa tecnica come alternativa alla replicazione, in particolare supporta due tipologie di pool: replicated ed erasure-coded. I pool erasure-coded riducono la quantità di spazio su disco richiesta per assicurare la durabilità dei dati, ma risultano computazionalmente più costosi di quelli che adottano la replicazione. Questo rende l'erasure coding vantaggioso quando lo storage dei dati deve essere durevole e resistente ai guasti, ma non richiede elevate performance in lettura.

Entrando più nel dettaglio di come funziona e come venga utilizzato in Ceph, vediamo come vengono effettuate letture e scritture in RADOS utilizzando l'erasure coding, facendo riferimento all'esempio nelle Figure 8A (scrittura) e 8B (lettura).

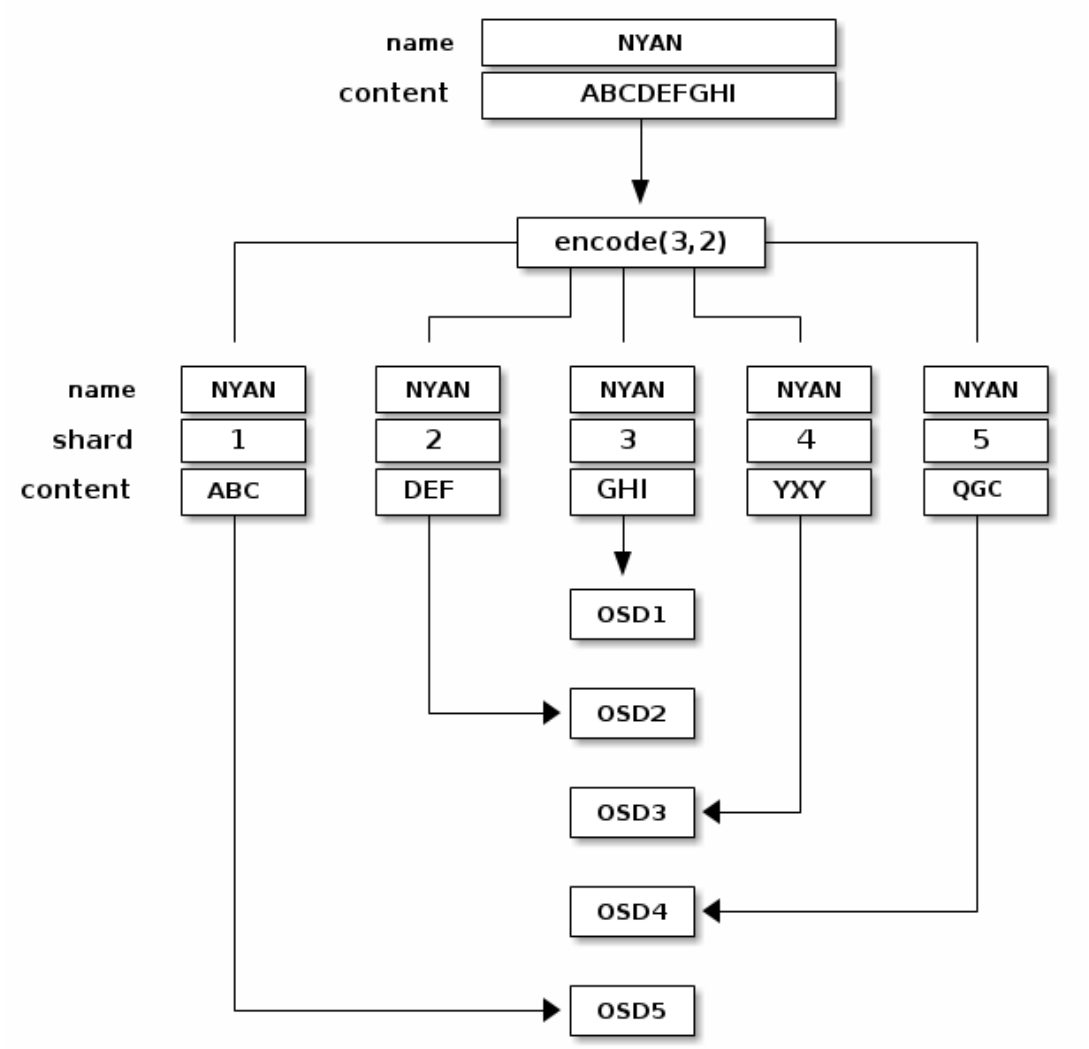

Figura 8A - Esempio di scrittura con erasure coding

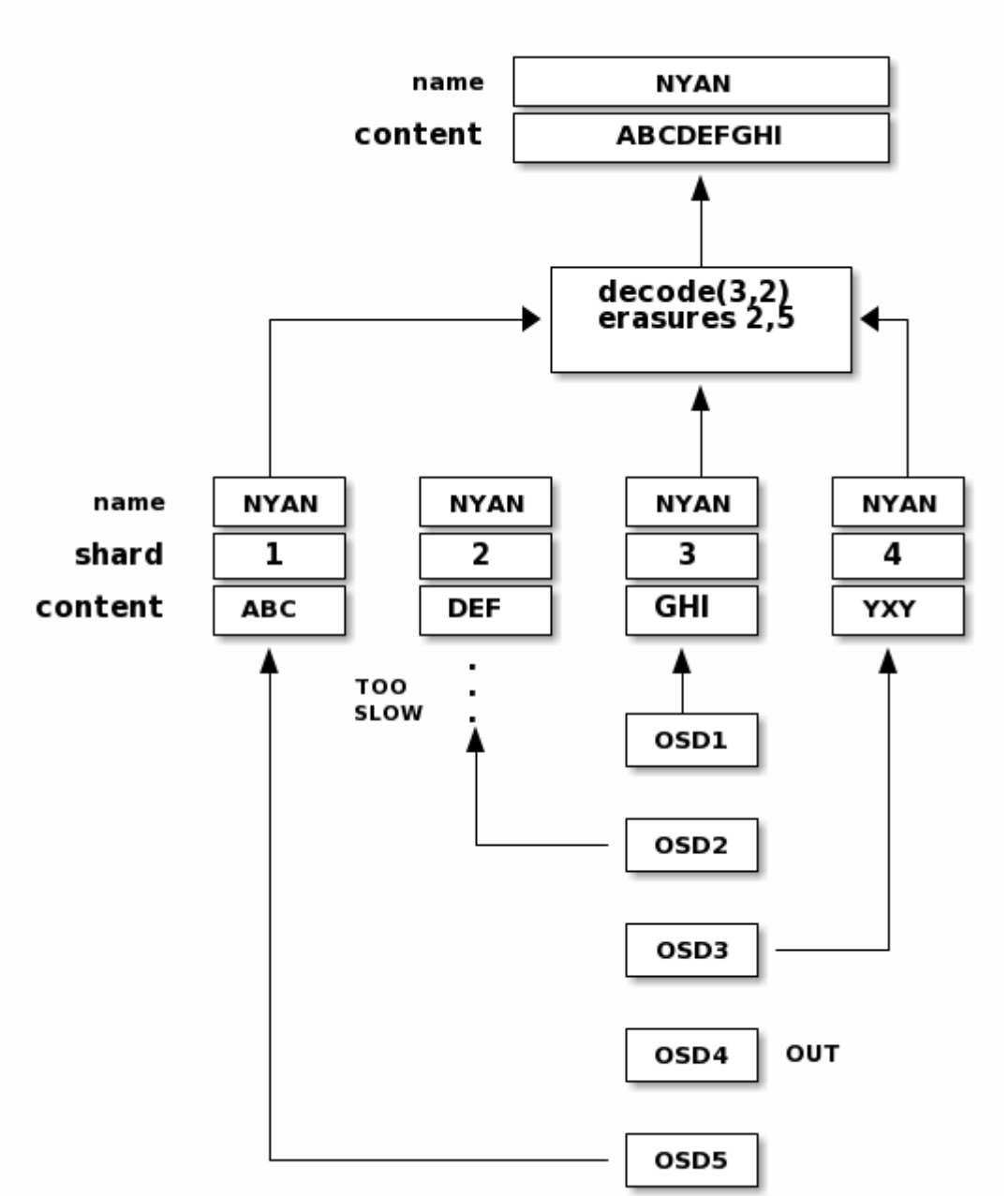

Figura 8B - Esempio di lettura con erasure coding

La Figura 8A mostra come un oggetto NYAN, contentente i dati ABCDEFGHI, viene suddivisa in tre chunk dati (i cui rispettivi contenuti sono ABC, DEF e GHI) e due chunk di erasure-coding, ognuno dei quali verrà salvato in un diverso OSD. Quando un oggetto deve essere letto da un pool, la funzione di decodifica dovrà recuperare i chunk dell'oggetto per poi decodificarli, questa operazione può essere effettuata nonappena tre chunk vengono letti, ed attraverso questa sarà quindi possibile ricomporre il contenuto originale dell'oggetto. Nella situazione in cui OSD 4 dovesse fallire e OSD 2 dovesse essere troppo lento, come possiamo vedere nella Figura 8B,

verrebbbero recuperati i tre chunk seguenti: shard 1 contenente ABC, shard 3 che contiene GHI e shard 4 con YXY; una volta letti questi sarà possibile ricostruire l'oggetto iniziale utilizzando la funzione di decoding.

1.4.5. Consistenza forte

RADOS è progettato per fornire un alto livello di consistenza, in particolare è categorizzato come sistema CP all'interno del teorema CAP, garantendo quindi consistenza e partition tolerance a scapito dell'availability; in questa sezione entreremo nel dettaglio di come viene garantita tale consistenza. Come già accennato precedentemente, tutti i messaggi di RADOS, che siano provenienti dai client o da altri OSD, sono taggati con l'epoca della mappa più recente conosciuta in modo da assicurare che tutte le operazioni siano applicate in modo interamente consistente. Se un client invia un'operazione di I/O ad un OSD errato a causa di una mappa non aggiornata, questo OSD risponderà con gli aggiornamenti necessari a mettere in pari il client, che una volta aggiornata la mappa ridirigerà la richiesta verso il destinatario corretto. Questo evita il bisogno di condividere la mappa in modo proattivo con i client, che verranno notificati della presenza di aggiornamenti non appena interagiranno con lo storage cluster. In molti casi impareranno di aggiornamenti che non avranno effetti sull'operazione corrente ma in ogni caso permetterà di rindirizzare le richieste future in modo corretto. Se viene aggiornata la master copy della mappa del cluster (quella gestita dai monitor) relativa ad un PG, i vecchi membri possono elaborare le operazioni relative a questo PG se non sono ancora a conoscenza di tale modifica della mappa. Appena un OSD verrà a conoscenza dell'aggiornamento della mappa lo propagherà al primo scambio di messaggi con i suoi peer. Nel caso in cui il primo OSD ad apprendere la modifica della mappa sia uno contenente una replica di un PG (non il primario) e venga richiesta un'operazione all'OSD primario ancora inconsapevole del cambiamento, quest'ultimo proverà a propagare tale modifica ma riceverà in risposta il nuovo incremento della mappa. Questo è completamente sicuro perché qualsiasi gruppo di OSD che è il nuovo responsabile di un PG è tenuto a contattare tutti i nodi precedentemente responsabili al fine di determinare il contenuto corretto del PG; ciò garantisce che gli OSD precedenti apprendano la modifica e interrompano l'esecuzione dell'operazione I/O prima dell'avvio dei nuovi OSD responsabili. Ottenere una consistenza per le operazioni di lettura simile a quella delle scritture è leggermente più complesso. In caso di guasto di rete che fa sì che un OSD diventi solo parzialmente irraggiungibile, esso potrebbe essere dichiarato "failed" ma rimanere comunque raggiungibile dai client aventi una mappa non aggiornata, mentre quella aggiornata potrebbe specificarne uno nuovo al suo posto. Per ovviare a questo problema vengono bloccate tutte le letture da ogni OSD che non riceve heartbeat da suoi pari per H secondi. Per evitare che qualsiasi operazione di lettura venga elaborata dal vecchio OSD dopo che sono state effettuate scritture da quello nuovo, è necessario utilizzare heartbeat frequenti tra gli OSD per ogni PG, in modo tale che questi rimangano leggibili. Prima che un altro OSD assuma il ruolo di primario per un PG, deve ottenere un riconoscimento positivo dal vecchio OSD (assicurandosi che sia consapevole del cambio di ruolo) o ritardare tale azione per lo stesso intervallo di tempo scelto per gli heartbeat. Nell'attuale implementazione viene utilizzato un intervallo di heartbeat relativamente breve (due secondi) così da garantire sia il rilevamento tempestivo dei guasti che un breve intervallo di indisponibilità dei dati dei PG in caso di fallimento dell'OSD primario.

1.4.6. Migrazione dei dati, rilevazione e recupero dai fallimenti

In RADOS i meccanismi di migrazione dei dati e di recupero dai fallimenti sono guidati interamente dagli aggiornamenti della mappa del cluster e dalle successive modifiche

nella mappatura dei placement group agli OSD. Tali cambiamenti possono essere dovuti al fallimento di device, al loro recupero, ad una espansione o riduzione del cluster o anche ad un completo rimescolamento dei dati dovuto all'impiego di una nuova politica di distribuzione CRUSH. RADOS impiega un robusto algoritmo di peering per stabilire una vista consistente dei contenuti dei PG tra gli OSD nel quale sono allocati e ripristinare la giusta distribuzione e replicazione dei dati. Questa strategia si basa sulla premessa che gli OSD replicano in modo aggressivo il log di un PG ed il registro di ciò che esso attualmente contiene (ad esempio quali sono le versioni degli oggetti al suo interno), anche quando le repliche di oggetti potrebbero mancare localmente. Pertanto, anche se il ripristino è lento e la sicurezza degli oggetti viene ridotta per un certo periodo di tempo, i metadati dei PG vengono protetti con cura, semplificando l'algoritmo di ripristino e consentendo al sistema di rilevare in modo affidabile la perdita di dati. Quando un OSD riceve un aggiornamento della mappa del cluster, percorre tutti i nuovi incrementi fino al più recente per esaminare e possibilmente correggere i valori di stato del PG. Ogni PG immagazzinato in locale, la cui *active list* di OSD cambia, viene marchiato e deve effettuare nuovamente il peering. Considerare tutte le epoche delle mappe (e non solo la più recente) garantisce che siano prese in considerazione anche le distribuzioni di dati intermedie, in modo tale che se un OSD viene rimosso da un PG ed aggiunto di nuovo, è possibile rendersi conto che possono essere stati fatti aggiornamenti intermedi ai contenuti del PG. Il meccanismo di peering, e successivamente quello di recovery, procedono indipendentemente per ogni PG del sistema e vengono gestiti dall'OSD primario di ogni PG. Ogni OSD invia un messaggio di notifica al primario corrente quando una replica del PG viene immagazzinata al suo interno. Tale messaggio include informazioni base sullo stato del PG locale tra cui: l'aggiornamento più recente, i limiti del log del PG e l'epoca più recente conosciuta durante la quale il PG ha effettuato il peering con successo. I messaggi di notifica garantiscono che, al variare di una mappa nel quale cambia l'OSD primario, questo nuovo dispositivo scopra di essere diventato il primario senza contattare direttamente tutti i PG, in quanto la prima replica con la nuova mappa invierà il messaggio di notifica con l'epoca aggiornata. L'OSD primario genererà un *prior set*, cioè l'insieme dei dispositivi che potrebbero aver partecipato per il PG in questione dall'inizio dell'epoca precedente; dato che è un limite inferiore, può essere aggiustato nel tempo nel caso vengano ricevute altre notifiche (riducendo così il prior set). Questo prior set può essere richiesto esplicitamente per evitare un'attesa indefinita per OSD aventi una mappa precedente che non memorizzano effettivamente il PG (ad esempio, se il peering non è mai stato completato per una mappatura PG intermedia). Dato che l'OSD primario possiede i metadati del PG per l'intero prior set, può determinare l'aggiornamento più recente applicato a qualunque replica, e richiedere qualsiasi frammento di log necessario dagli OSD precedenti per sincronizzare i log del PG sulle repliche attualmente attive. Qualora i log disponibili fossero insufficienti (ad esempio nel caso uno o più OSD non avessero dati sul PG in questione) verrebbe generata una lista con l'intero contenuto del PG; questo non risulta necessario per riavvii o brevi blackout, in quanto i log più recenti del PG sono sufficienti per sincronizzare nuovamente le repliche. Infine l'OSD primario condividerà i frammenti dei log mancanti con gli OSD di replica, così che tutti possano conoscere gli oggetti che il PG dovrebbe contenere (anche nel caso siano ancora mancanti in locale), e inizia a processare le operazioni di I/O mentre in background avviene il processo di recovery.

Per la comunicazione fra componenti, RADOS utlizza una libreria per lo scambio di messaggi asincrono, ordinato e point to point. Nel caso di un eventuale fallimento della comunicazione, verrà effettuato un numero limitato di tentativi di riconnessione, prima di riportare l'avvenuto fallimento al cluster di monitor. I nodi

di storage scambiano periodicamente messaggi di *heartbeat* con i loro peer, in modo tale da garantire la rilevazione di eventuali fallimenti di dispositivi. Gli OSD che scoprono di essere stati marchiati come *down* sincronizzano il proprio storage locale e si terminano per garantire un comportamento consistente. Un vantaggio fondamentale della replicazione declusterizzata è la capacità di parallelizzare il ripristino in caso di fallimento. Le repliche condivise con un dispositivo *failed* vengono distribuite su molti altri OSD ed ogni PG sceglierà indipendentemente un sostituto, permettendo la replicazione sullo stesso numero di OSD. Sebbene ogni OSD, possedendo i metadati dei PG, è in grado di recuperare indipendentemente gli oggetti mancanti, questo non viene effettuato in quanto questa strategia presenta due limitazioni. Prima di tutto, il fatto che multipli OSD recuperino indipendentemente gli oggetti dallo stesso PG farebbe in modo che non operino in modo sincronizzato ripetendo tutti le stesse azioni ma in momenti diversi e su oggetti diversi anche dallo stesso OSD, portando quindi alla duplicazione dell'aspetto più costoso del recupero: la ricerca e la lettura dei dati. Inoltre i protocolli di replicazione RADOS diventano sempre più complessi se agli OSD di replica mancano gli oggetti modificati. Per ovviare a queste limitazioni, il recupero dei PG in RADOS è quindi coordinato interamente dall'OSD primario. Le operazioni sugli oggetti mancanti sono rimandate fino a quando il primario non ne possiede una copia in locale. Grazie al processo di peering, l'OSD primario è a conoscenza degli oggetti mancanti ad ogni replica e può effettuare delle *push* preventive per fornirgli tali oggetti, che stanno per essere modificati, semplificando così la logica di replicazione e assicurandosi che gli oggetti vengano letti una volta sola. Se il primario sta eseguendo una *push* (ad esempio in risposta ad una richiesta di pull effettuata da una replica) o ha effettuato una *pull* di un oggetto, propagherà sempre tale oggetto a tutte le repliche che ne necessitano una copia fino a quando l'oggetto sarà presente in memoria; in questo modo ogni oggetto replicato nuovamente verrà letto una volta sola.

1.4.7. Paxos

Come visto precedentemente, per il funzionamento di RADOS sono necessari due compomonenti fondamentali: gli OSD ed i monitor; mentre gli OSD si occupano della memorizzazione e gestione dei dati, i monitor formano un cluster e lavorando in modo coordinato mantenengono e gestiscono la copia master dello storage cluster, aggiornandola periodicamente in risposta a cambiamenti nella configurazione del sistema o a cambi di stato degli OSD (come ad esempio fallimento o ripristino di un dispositivo). Il cluster dei monitor è basato sul concetto di macchina a stati distribuita, dove la mappa rappresenta lo stato corrente ed ogni aggiornamento porta ad una nuova epoca; per garantire che tutti i monitor condividano la stessa versione della mappa del cluster viene utilizzato l'algoritmo di consenso Paxos, in modo da fornire consistenza a discapito della disponibilità e della latenza degli aggiornamenti. Per effettuare operazioni come la lettura o l'aggiornamento della mappa del cluster è richiesto che la maggior parte dei monitor sia disponibile. La versione di Paxos utilizzata dai monitor è leggermente semplificata rispetto l'originale, in questa infatti viene permessa la proposta di un singolo aggiornamento per volta, rendendo più semplice la sua implementazione, e gli aggiornamenti vengono coordinati tramite un meccanismo di emissione di *lease* per garantire la consistenza dell'ordine delle operazioni di lettura ed aggiornamento della mappa del cluster. Il cluster inizialmente elegge un leader che avrà il compito di serializzare gli aggiornamenti della mappa e gestirne la consistenza; una volta eletto, il leader richiederà ad ogni monitor l'epoca delle mappe in loro possesso. Questi avranno a disposizione un quantitativo fisso di tempo per rispondere a tale richiesta (di default due secondi) e di conseguenza unirsi al quorum. Il leader si assicurerà che la maggioranza dei

monitor sia attiva e che possiedano l'epoca più recente della mappa del cluster (richiedendo update incrementali agli altri monitor per sincronizzarsi, qualora avessero un'epoca precedente), a quel punto inizierà a distribuire *lease* di breve durata ai monitor attivi. Ogni lease fornisce il permesso di distribuire copie della cluster map ad ogni OSD o client che ne faccia richiesta; questo deve essere rinnovato dal leader prima che la sua durata termini e ad ogni ricezione, ogni monitor dovrà rispondere con un rapido acknowledge. Verrà indetta una nuova elezione nei seguenti casi:

- un lease non viene rinnovato prima che la sua durata termini (si assume che il leader sia terminato);
- a seguito dell'invio di un lease, il leader non riceve l'acknowledge dal monitor entro un certo intervallo di tempo (si assume che il monitor sia terminato);
- un monitor si avvia per la prima volta;
- un'elezione chiamata in precedenza non viene completata entro un intervallo di tempo ragionevole.

Quando un monitor attivo riceve una richiesta di aggiornamento (ad esempio in seguito alla segnalazione di un fallimento da parte di uno degli OSD), per prima cosa controlla se la richiesta ricevuta è nuova, in questo caso la inoltra al leader che la prenderà in carico aggregandola agli altri aggiornamenti, mentre in caso contrario può semplicemente rispondere con gli aggiornamenti incrementali necessari per sincronizzare il segnalante. Nel caso il leader riceva la richiesta di aggiornamento da un monitor, la aggregherà e la serializzerà con gli aggiornamenti, poi periodicamente incrementerà l'epoca della mappa del cluster e, tramite i protocolli di aggiornamento di Paxos, distribuirà gli aggiornamenti agli altri monitor revocando simultaneamente i lease emessi in precedenza. Quando il leader avrà ricevuto gli acknowledge degli aggiornamenti dalla maggior parte dei monitor, invierà un ultimo messaggio contenente una nuova serie di lease. Tramite queste operazioni è garantito quindi che ogni lease scada prima che venga eseguito un aggiornamento della mappa (per la revoca da parte del leader o per la terminazione della sua durata); ne consegue che ogni sequenza di aggiornamenti e letture della mappa risulteranno in versioni progressive (l'epoca può solo crescere), a prescindere da quale sia il monitor a cui vengono inviati i messaggi, a condizione che sia disponibile la maggior parte dei monitor.

1.5. I client RADOS

Ceph fornisce tre tipi di client con i quali è possibile manipolare oggetti in RADOS: Rados Block Device (RBD), Rados Gateway (RGW o RadosGW) e Ceph File System (CephFS). Ognuno di questi client funge da interfaccia verso gli utenti convertendo i dati dalla rappresentazione dell'utente a quella utilizzata in RADOS e viceversa; in particolare RBD fornisce un'immagine del dispositivo a blocchi, RGW una rappresentazione sotto forma di oggetti RESTful e CephFS un'organizzazione di file e cartelle in una struttura ad albero. Infine vi è un ulteriore modo per manipolare i dati in RADOS, la libreria Librados; questa consente di interfacciarsi direttamente con RADOS aumentando notevolmente le performance, ma portando gli svantaggi di dover lavorare con gli oggetti a basso livello ed effettuare tutte le operazioni che già vengono fornite di base dai client. Gli altri client sono basati su questa libreria per fornire le loro funzionalità ed interagire con RADOS; sfruttando le API fornite da Librados, ogni utente è in grado di creare un client custom, adatto alle proprie necessità, per interfacciarsi con lo storage cluster. Questa libreria è implementata in diversi linguaggi di programmazione consentendo agli utenti di utilizzare le sue API in C,

C++, Java, Python, Ruby o PHP; attraverso queste è possibile interagire con entrambi i componenti fondamentali di RADOS: con i monitor per richiedere la master copy della mappa del cluster e con gli OSD per leggere e scrivere oggetti nei nodi di storage.

I dispositivi di archiviazione hanno limitazioni di velocità che influiscono sulle prestazioni e sulla scalabilità, per questo motivo spesso supportano lo striping, ovvero una tecnica che prevede la divisione dei dati in più parti per poi archiviarli in modo sequenziale su più dispositivi, aumentando così la velocità effettiva, le prestazioni e favorendo la scalabilità. Ceph utilizza questa tecnica per dividere i dati in stripe ed allocarli negli oggetti che poi saranno salvati all'interno di RADOS; vi sono diversi modi per effettuare questa operazione, il più semplice prevede la scrittura di un numero di stripe in un oggetto fino a quando questo non raggiunge la sua capacità massima, per poi crearne un'altro e continuare l'operazione (vedi Figura 9A). Questa metodologia può essere sufficiente per gestire immagini di dispositivi a blocchi, oggetti e file system di dimensioni ridotte, anche se con prestazioni limitate, in quanto non sfrutta al massimo la capacità di Ceph di distribuire i dati tra i placement group. Per immagini, oggetti o file system di grandi dimensioni questa riduzione delle performance può diventare significativa; in questo caso è possibile migliorare le prestazioni utilizzando una metodologia diversa di striping, che prevede la divisione delle stripe su set di oggetti, come si può vedere in Figura 9B. Con questa tecnica è possibile ottenere un incremento notevole delle prestazioni parallelizzando le scritture degli stripe sugli oggetti appartenenti ad un set; dato che gli oggetti vengono mappati su diversi placement group per poi essere scritti su diversi OSD, ogni scrittura avviene in modo parallelo alla velocità massima di ogni disco. Nell'esempio in Figura 9B abbiamo set di oggetti di dimensione 4; i dati vengono sottoposti a striping e divisi sugli oggetti del primo set in modo parallelo, quindi il primo stripe verrà messo nel primo oggetto e così via fino al quarto che sarà allocato nel quarto oggetto. Dopo che il quarto stripe è stato scritto viene controllato se il set ha raggiunto la sua capienza massima, in caso contrario si continua la scrittura sul primo set altrimenti ne viene creato uno nuovo seguendo la stessa strategia di distribuzione.

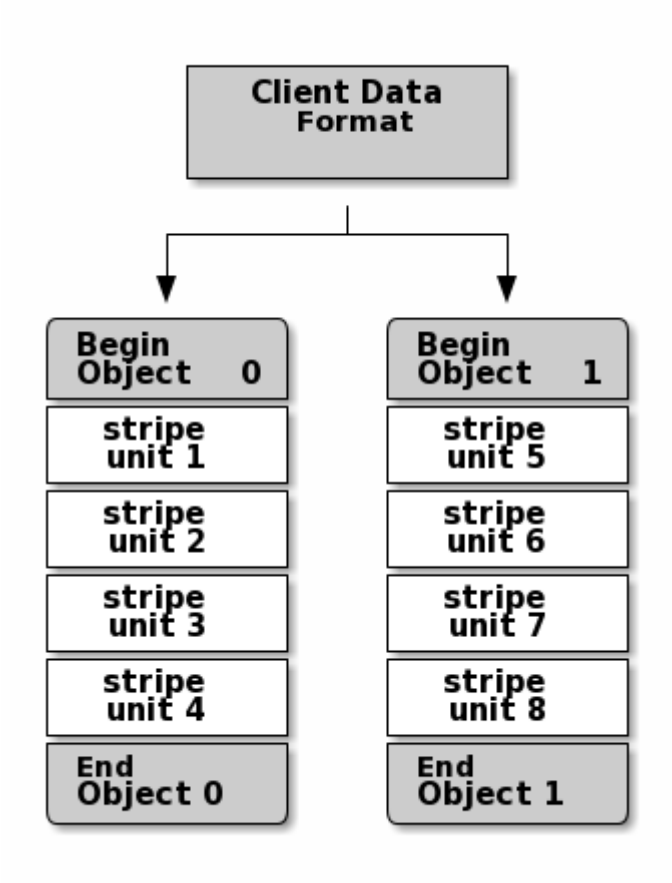

Figura 9A - Primo striping

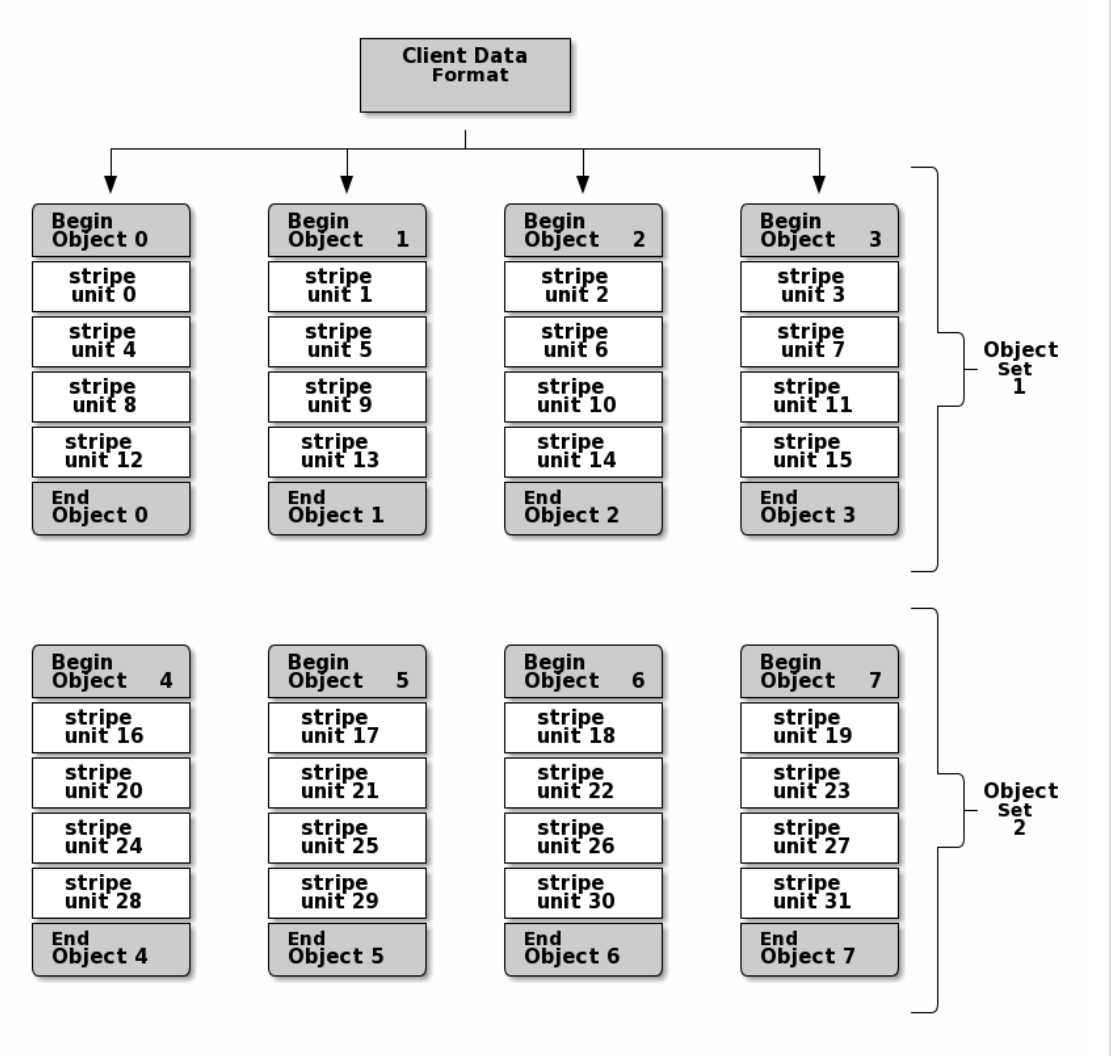

Figura 9B - Secondo striping

Lo striping dei dati in oggetti non viene effettuato da RADOS ma dai client RGW, RBD e CephFS; Librados non effettua questa operazione quindi i client che utilizzano tale libreria per operare direttamente sugli oggetti RADOS dovranno provvedere ad effetuarla autonomamente per ottenerne i benefici. È possibile scegliere i parametri di striping prima di portare un sistema in produzione in quanto una volta scritti oggetti, non sarà più possibile modificare tali parametri.

1.5.1. CephFS

Il Ceph File System, o CephFS, è un client che offre funzionalità di storage sotto forma di file system, conforme a POSIX e costruito sopra RADOS, per memorizzare dati all'interno dello storage cluster distribuito di Ceph. Ha come obiettivo quello di fornire un archivio di file all'avanguardia, multiuso, altamente disponibile e performante per una varietà di applicazioni, inclusi casi d'uso tradizionali come home directory condivise, spazio di lavoro HPC e distribuzione del carico di lavoro su memoria condivisa. Per far ciò sono state effettuate alcune nuove scelte architetturali, come la separazione in pool RADOS diversi tra i dati ed i metadati dei file, dove questi ultimi vengono gestiti, memorizzati e serviti tramite un cluster di

Metadata Server (MDS) ridimensionabile, che può quindi scalare per supportare carichi di lavoro dei metadati con throughput più elevato. Questa scelta è stata effettuata per alleggerire il carico degli OSD in quanto, operazioni semplici come elencare o cambiare cartella (ls o cd), andrebbero ad appesantire inutilmente questi ultimi. I servizi MDS hanno il compito di memorizzare tutti i metadati del file system, comprese la struttura, le proprietà dei file, le modalità di accesso, ecc... I client del file system possono interagire con due entità di Ceph: con gli OSD, per leggere e scrivere blocchi di dati di file avendo diretto accesso a RADOS, o con gli MDS per operare con i metadati. I dati dei file vengono mappati in uno o più oggetti ed archiviati dagli OSD nello storage cluster; dato che non è presente alcun gateway o broker che media le operazioni di I/O dei dati dei file e che i client hanno accesso diretto a RADOS, i carichi di lavoro dei dati dei file sono in grado di scalare linearmente con la dimensione dello storage cluster sottostante. Gli MDS non memorizzano lo stato dei metadati localmente, ma aggregano tutte le mutazioni dei metadati attraverso una serie di scritture su un journal memorizzato in RADOS; questi coordinano l'accesso ai dati e fungono da autorità per lo stato della cache distribuita dei metadati (journal) mantenuta in modo cooperativo da client ed MDS. Questo modello consente agli MDS una collaborazione rapida e coerente. Ogni istanza MDS può assumere due possibili stati:

- attivo, utili per aumentare la scalabilità. Gli MDS in questo stato saranno operativi e si suddivideranno l'albero delle cartelle, bilanciando efficacemente il carico di lavoro; almeno uno deve essere in questo stato per utilizzare la funzionalità CephFS.
- standby, utili per aumentare la disponibilità. Gli MDS in questo stato non sono operativi, ma sono pronti ad assumere i compiti di qualsiasi MDS nel caso questo passasse da attivo a guasto a causa di un fallimento. Questa transizione viene attivata automaticamente dai monitor.

È possibile avere diversi MDS distribuiti su più macchine fisiche e combinando i loro stati è possibile ottenere le caratteristiche desiderate, ad esempio potrebbe essere utile avere più istanze attive per avere un buon livello di scalabilità e un'istanza standby per aumentare la disponibilità. CephFS è la più vecchia interfaccia di archiviazione in Ceph e una volta era il metodo principale per utilizzare RADOS; ora è affiancato dalle altre due interfacce per formare un sistema di archiviazione moderno ed unificato.

1.5.2. RADOS Gateway

Rados Gateway, RadosGW o RGW, è un servizio FastCGI costruito sopra Librados che fornisce API RESTful per leggere o memorizzare oggetti e metadati nello storage cluster, sfruttando interfacce compatibili con (Amazon) S3 e (OpenStack) Swift. È possibile pensare ad RGW come uno strato che si sovrappone allo storage cluster, che utilizza il proprio formato dei dati, e dispone di una propria gestione degli utenti, mantenendo un loro database ed un sistema di autenticazione e di controllo degli accessi. Il gateway RADOS utilizza uno spazio dei nomi comune, ciò significa che è possibile utilizzare le API compatibili con Swift o con S3 in modo intercambiabile sugli stessi dati, ad esempio è possibile scrivere dati utilizzando le API S3 con un'applicazione e leggerli tramite un'altra applicazione che utilizza Swift. Il termine oggetto viene utilizzato sia in RADOS che in S3 e Swift, ma sono due concetti separati e non sono intercambiabili; in Ceph, il termine oggetto viene utilizzato per descrivere i dati che archivia, qualsiasi sia l'interfaccia dal quale questi siano prodotti, mentre in S3 e Swift viene utilizzato per identificare un insieme di dati che un utente vuole memorizzare. Gli oggetti RADOS non sono gli stessi di S3 e Swift nel caso si utilizzi l'interfaccia RadosGW; un oggetto S3 o Swift non corrisponde

necessariamente ad un oggetto RADOS, ma può essere mappato dal client in uno o più di questi. RadosGW supporta la configurazione a multisito che permette di suddividere lo storage cluster in più siti così da permettere la replicazione dei cluster in locazioni fisiche differenti; questa caratteristica permette di organizzare il cluster in modo da allocare repliche distribuite geolocalmente, aumentando la latenza tra questi siti, ma evitando così il fallimento dell'intero cluster. In questo contesto è necessario definire i concetti di:

- realm, rappresenta un insieme di zonegroup che condividono uno spazio dei nomi univoco a livello globale per tutti gli oggetti ed i bucket nello spazio multisito, inoltre contiene la configurazione e lo stato del multisito. Il reame contiene i gruppi di zone e solo una di esse deve essere il zonegroup master;
- zonegroup, rappresenta un raggruppamento di zone dove deve essere presente una ed una sola zone master e le altre sono secondarie. Nel caso di una zonegroup con due zone, una master ed una secondaria, i dati saranno replicati dalla master all'altra;
- zone, rappresenta una o più istanze RadosGW eseguite nello stesso cluster RADOS.

Sono supportate diverse opzioni di configurazione multisito possibili, ad esempio:

- zona singola, si ha una sola zona composta da una o più istanze di RadosGW che puntano ad un singolo storage cluster, in modo da bilanciare il carico delle richieste del client;
- multi-zone: si ha un zonegroup e più zone, ciascuna zona supportata dal proprio storage cluster e con una o più istanze RadosGW in esecuzione. Avere più zone in un zonegroup permette il "disaster recovery" cioè la possibilità di ripristinare lo storage cluster nel caso si verificasse un errore significativo come il fallimento dell'intera zona. Inoltre, ogni zona è attiva e può ricevere operazioni di lettura e scrittura, permettendo di ridurre i tempi per interagire col sistema per i client vicini alla seconda zona;
- multi-zonegroup: si hanno più zonegroup all'interno dello stesso realm, ognuna con una o più zone. Essendo nello stesso reame condivideranno lo stesso spazio dei nomi, garantendo quindi che gli oggetti abbiano identificativi univoci tra le diverse zone ed i diversi zonegroup;
- multi-realm: avere più realm fornisce la capacità di supportare molteplici configurazioni e spazi dei nomi.

1.5.3. RADOS Block Device

Rados Block Device, o RBD, è il client più comunemente utilizzato ed offre funzionalità di storage di dispositivi a blocchi virtuali permettendo di memorizzare immagini di dischi nello storage cluster. Il sistema di archiviazione su dispositivi a blocchi è quello più maturo e diffuso per memorizzare dati su supporti come HDD, SSD, CD o altri; un blocco è una sequenza di byte di lunghezza fissa e la combinazione di molti blocchi in un unico file può essere utilizzata come dispositivo di archiviazione da cui leggere e scrivere. RBD fornisce funzionalità di archiviazione di dispositivi a blocchi ridimensionabili e, poiché è costruito su librados, eredita e sfrutta le funzionalità e le caratteristiche di RADOS, come ad esempio la possibilità di effettuare snapshot in sola lettura ed effettuare ripristini con essi, l'allocazione con thin provisioning, la clonazione di immagini, la replicazione e la consistenza forte. I client di tale interfaccia comunicano con lo storage cluster tramite i moduli kernel o tramite la libreria librbd utilizzata dai sistemi hypervisor, evitando il sovraccarico degli oggetti del kernel per i sistemi virtualizzati. RBD esegue lo

striping delle immagini dei dispositivi a blocchi in diversi oggetti, distribuibuendoli su più placement group allocati in OSD sparsi in tutto il cluster, questo consente di ottenere prestazioni elevate anche per immagini di grandi dimensioni. RBD con snapshot e thin-provisioning sono un'opzione interessante per la virtualizzazione ed il cloud computing, infatti offre prestazioni elevate con un'ampia scalabilità alle Kernel Virtual Machine (KVM) come ad esempio Quick Emulator (QEMU), ed ai sistemi di elaborazione cloud-based come OpenStack e CloudStack, che si basano su libvirt e QEMU per l'integrazione con i dispositivi a blocchi Ceph. Negli scenari che utilizzano macchine virtuali, solitamente viene effettuato il deploy di un Ceph block device con il driver di archiviazione di rete rbd in QEMU/KVM, dove la macchina host utilizza librbd per fornire un servizio di archiviazione di dispositivi a blocchi al guest, disaccoppiando così lo storage dall'host. Per integrare tecnologie di virtualizzazione per le quali non viene ancora fornito il supporto per librbd, è possibile utilizzare lo strumento a linea di comando rbd che, sfruttando i moduli kernel, permette di interagire con l'interfaccia RBD come client. Dato che RADOS Block Device si interfaccia con lo stesso storage cluster che fornisce le altre interfacce, è possibile operare contemporaneamente utilizzando anche RadosGW e CephFS. Utilizzare RBD ha diversi benefici e porta alcune proprietà interessanti come:

- è possibile ridimensionare le immagini, clonarle e rinominarle
- le immagini vengono assegnate in modalità thin-provisioning
- è possibile importare ed esportare immagini
- è possibile effettuare snapshot in sola lettura ed effettuare ripristini da questi
- è possibile utilizzare i moduli kernel o la libreria librbd

2. Setup del cluster ed esempi

In questa sezione descriviamo le macchine che abbiamo utilizzato ed il procedimento per effettuare il setup del cluster ed i test eseguiti per verificare le principali funzionalità di Ceph.

2.1. Preparazione macchine

- Creare una macchina virtuale
  - 4 GB RAM
  - 50 GB di disco
- Installare Ubuntu server 20.04.1 aggiungendo Openssh-server nell'installazione
  - nome utente: ceph
  - nome macchina: ceph-00
  - password: 123
- Aggiungere un secondo disco da 15 GB
- (facoltativo: se si vuole accedere tramite ssh dalla macchina host) Aggiungere la regola di port forwarding per ssh (porta 22) alla scheda di rete con NAT, nel nostro caso abbiamo mappato la porta 22 del guest alla porta 1022 dell'host
- Aggiungere una seconda scheda di rete come "Rete interna" chiamata cephnet
- Avviare la macchina virtuale e configurare la rete interna
  - Creare la configurazione con la seguente istruzione:

```
sudo nano /etc/netplan/00-installer-config.yaml
```

- Copiare questa configurazione nel file:

```
network:
ethernets:
enp0s3:
dhcp4: true
enp0s8:
dhcp4: false
addresses: [192.168.1.20/24]
gateway4: 192.168.1.255
nameservers:
addresses: [8.8.8.8,8.8.4.4]
version: 2
```

- Applicare la configurazione

```
sudo netplan --debug apply
```

- Controllare l'indirizzo della macchina usando il comando ifconfig (installarlo se necessario con la seguente istruzione)

```
sudo apt install net-tools
```

- Aggiornare tutto

```
sudo apt update
sudo apt upgrade
```

- Abilitare l'account root per ssh
  - Modificare la configurazione di ssh:

```
sudo nano /etc/ssh/sshd_config
```

  - Togliere il commento alla linea commentata seguente e sostituirla con:

```
PermitRootLogin yes
```

  - Riavviare il servizio sshd

```
sudo systemctl restart sshd
```

  - Modificare la password di root con il comando:

```
sudo passwd root
```

- Passare all'utente root

```
sudo su root
```

- Assicurarsi di aver installato python3

```
python3 --version
```

nel nostro caso la versione utilizzata è la 3.8.5

- Installare ntp e selezionare la timezone giusta

```
sudo apt update
sudo apt install ntp
sudo timedatectl set-timezone Europe/Rome
```

- Modificare gli hosts
  - Aprire il file /etc/hosts

    ```
    sudo nano /etc/hosts
    ```

  - Aggiungere i seguenti host ```

192.168.1.20 ceph-00 192.168.1.21 ceph-01 192.168.1.22 ceph-02 192.168.1.23 ceph-03 ```

- Installare docker-ce

```
sudo apt-get install apt-transport-https ca-certificates curl software-properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add –
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
sudo apt update
sudo apt-get install docker-ce
sudo systemctl start docker
sudo systemctl enable docker
```

- Scaricare ed installare cephadm

```
curl --remote-name --location https://github.com/ceph/ceph/raw/octopus/src/cephadm/cephadm
chmod +x cephadm
./cephadm add-repo --release octopus
./cephadm install
```

- Verificare la corretta installazione

```
which cephadm
```

che dovrà ritornare /usr/sbin/cephadm

- Riavviare e installare ceph-common

```
apt install ceph-common
```

- Aprire la porta 8443 nella scheda di rete NAT
- Clonare la macchina in altre 3 chiamate ceph-01, ceph-02 e ceph-03 e per ognuna
  - Modificare le porte di forwarding da VirtualBox

La nostra configurazione è la seguente

Macchina	Porta host	Porta guest
ceph-00	1022	22
ceph-00	8443	8443
ceph-01	1023	22
ceph-01	8444	8443
ceph-02	1024	22
ceph-02	8445	8443
ceph-03	1025	22
ceph-03	8446	8443

- Modificare l'indirizzo

  ```
  sudo nano /etc/netplan/00-installer-config.yaml
  ```

  Per applicare la modifica:

  ```
  sudo netplan --debug apply
  ```

- Modificare l'hostname

  ```
  sudo hostnamectl set-hostname ceph-01
  ```

- Modificare il nome dell'host per l'indirizzo 127.0.1.1 con il nome della macchina relativa all'interno del file /etc/hosts

  ```
  nano /etc/hosts
  ```

modificare la seguente riga con il nome dell'host corretto: 127.0.1.1 ceph-00

```
- Generare e condividere la chiave ssh (compreso ceph-00)
- Generare la chiave
```

ssh-keygen

```
- Condividere la chiave con gli altri nodi (queste istruzioni sono quelle
eseguite dal guest ceph-00)
```

ssh-copy-id root@ceph-01 ssh-copy-id root@ceph-02 ssh-copy-id root@ceph-03

```
### 2.2. Bootstrap del cluster
```

- Eseguire il bootstrap utilizzando cephadm dal nodo ceph-00

```
cephadm bootstrap --mon-ip 192.168.1.20 --initial-dashboard-user admin --
initial-dashboard-password 123 --dashboard-password-noupdate
```

Alla fine del processo di bootstrap si avrà un cluster con un unico host su cui sono presenti un servizio "manager" ed un "monitor", più altri servizi di supporto come prometheus e grafana. Sarà inoltre possibile accedere alla dashboard da un browser della macchina host all'indirizzo: http://192.188.56.1:8443. Le credenziali saranno quelle inserite in precedenza nella fase di bootstrap.

2.3. Aggiunta dei Monitor

- Dall'host ceph-00 configurare gli indirizzi (della rete interna) degli host che ospiteranno gli altri monitor

```
ceph config set mon.ceph-01 public_addr 192.168.1.21
ceph config set mon.ceph-02 public_addr 192.168.1.22
ceph config set mon.ceph-03 public_addr 192.168.1.23
```

- Condividere la chiave pubblica di ceph con gli altri host

```
ssh-copy-id -f -i /etc/ceph/ceph.pub root@ceph-01
ssh-copy-id -f -i /etc/ceph/ceph.pub root@ceph-02
ssh-copy-id -f -i /etc/ceph/ceph.pub root@ceph-03
```

- Aggiungere gli host sui quali risiederanno gli altri monitor al cluster

```
ceph orch host add ceph-01
ceph orch host add ceph-02
ceph orch host add ceph-03
```

Alla fine al cluster saranno quindi aggiunti altri 3 monitor ed un altro manager.

- Una volta che i monitor sono entrati nel quorum, generare il file di configurazione minimale e sostituirlo a quello attuale

```
ceph config generate-minimal-conf > /etc/ceph/ceph-temp.conf
mv /etc/ceph/ceph-temp.conf /etc/ceph/ceph.conf
```

- Copiare la configurazione ed il keyring sugli altri host

```
scp /etc/ceph/ceph.client.admin.keyring /etc/ceph/ceph.conf root@ceph-
01:/etc/ceph/
scp /etc/ceph/ceph.client.admin.keyring /etc/ceph/ceph.conf root@ceph-
02:/etc/ceph/
scp /etc/ceph/ceph.client.admin.keyring /etc/ceph/ceph.conf root@ceph-
03:/etc/ceph/
```

2.4. Aggiunta degli OSD

- Dal nodo ceph-00 fare il deploy degli osd sui tre nodi ceph-01, ceph-02 e ceph-03:

```
ceph orch daemon add osd ceph-01:/dev/sdb
ceph orch daemon add osd ceph-02:/dev/sdb
ceph orch daemon add osd ceph-03:/dev/sdb
```

2.5. Verifiche effettuate

2.5.1. Prova del Ceph Storage Cluster tramite la RADOS CLI

- Abilitare l'autoscale dei pg dei pool

```
ceph config set global osd_pool_default_pg_autoscale_mode on
```

- Creare un pool e assegnare un'applicazione

```
ceph osd pool create test-pool
ceph osd pool application enable test-pool test-rados
```

- Creare un file

```
nano hello.txt

scrivere qualcosa al suo interno e salvare il file
```

- Aggiungere il file al cluster

```
rados -p test-pool put ciao hello.txt
```

- Da un altro guest (ad esempio ceph-01) elencare gli oggetti presenti per verificare che sia presente l'oggetto "ciao" e scaricarlo verificandone il contenuto

```
rados -p <pool-name> ls
rados -p test-pool get ciao hello2.txt
nano hello2.txt
```

- Rimuovere l'oggetto dal cluster

```
rados -p test-pool rm ciao
```

2.5.2. Prova di Ceph File System (CephFS)

- Dal nodo ceph-00 creare il file system

```
ceph fs volume create test-fs
```

- Creare la cartella sulla quale montare il file system e montarlo

```
mkdir /mnt/mycephfs
mount -t ceph :/ /mnt/mycephfs -o name=admin
```

- Creare un file

```
nano /mnt/mycephfs/hello.txt

scrivere qualcosa al suo interno e salvare il file
```

- Da un altro guest (ad esempio ceph-01) creare la cartella e montare il file system

```
mkdir /mnt/mycephfs
mount -t ceph :/ /mnt/mycephfs -o name=admin
```

- Verificare la presenza del file e del suo contenuto

```
nano /mnt/mycephfs/hello.txt
```

2.5.3. Prova di Ceph Block Device (RBD)

- Creare un pool ed inizializzarlo per RBD

```
ceph osd pool create block-pool
rbd pool init block-pool
```

- Creare l'immagine del dispositivo a blocchi

```
rbd create test-image --size 2048 --pool block-pool
```

- Mappare l'immagine

```
rbd map test-image -p block-pool
```

- Formattare l'immagine

```
mkfs.ext4 /dev/rbd0
```

- Montare l'immagine e scrivere un file

```
mkdir /mnt/mycephbd
mount /dev/rbd0 /mnt/mycephbd
nano /mnt/mycephbd/hello.txt

scrivere qualcosa al suo interno e salvare il file
```

- Smontare l'immagine

```
umount /mnt/mycephbd
```

- Da un altro guest (ad esempio ceph-01) mappare l'immagine, montarla e verificare presenza e contenuto del file

```
rbd map test-image -p block-pool
mkdir /mnt/mycephbd
mount /dev/rbd0 /mnt/mycephbd
nano /mnt/mycephbd/hello.txt
```

2.5.4. Prova di Ceph Object Gateway (RadosGW)

- Creare un reame

```
radosgw-admin realm create --rgw-realm=reame --default
```

- Creare un zonegroup

```
radosgw-admin zonegroup create --rgw-zonegroup=eu --rgw-realm=reame --master --default
```

- Creare una zona

```
radosgw-admin zone create --rgw-zone=it --rgw-zonegroup=eu --rgw-realm=reame --master --default
```

- Aggiornare il period

```
radosgw-admin period update --rgw-realm=reame --commit
```

- Mappare la porta 80 del guest su una porta dell'host (noi abbiamo utilizzato la 7080)
- Fare il deploy del gateway

```
ceph orch apply rgw reame it --placement="1 ceph-00"
```

- Creare un utente system per la dashboard ed iniettare le sue credenziali nella configurazione della dashboard

```
radosgw-admin user create --uid=system --display-name=system --system

copiare l'access-key e la secret-key ed inserirle nelle seguenti istruzioni:
ceph dashboard set-rgw-api-access-key <access_key>
ceph dashboard set-rgw-api-secret-key <secret_key>
```

- Riavviare il gateway

```
ceph orch restart rgw
```

A questo punto dovremmo poter vedere l'Object Gateway dalla dashboard

- Creare un altro utente e segnarsi le sue credenziali

```
radosgw-admin user create --uid=johndoe --display-name=johndoe

copiare l'access-key e la secret-key che saranno usate nello script python
```

- Testare il funzionamento eseguendo il seguente script python sostituendo access_key e secret_key con le credenziali ottenute nel passo precedente

```
import boto.s3.connection
access_key = '<USER-ACCESS-KEY>'
secret_key = '<USER-SECRET-KEY>'

print('\nConnessione come utente ')
conn = boto.connect_s3(
        aws_access_key_id = access_key,
        aws_secret_access_key = secret_key,
        host = '192.168.56.1',
        port = 7080,
        is_secure=False,
```

```
        calling_format = boto.s3.connection.OrdinaryCallingFormat(),
        )

print('\nCreazione bucket my-new-bucket')
bucket = conn.create_bucket('my-new-bucket')

print('\nCreazione oggetto ciao.txt')
key = bucket.new_key('ciao.txt')
key.set_contents_from_string('Hello World!')

print('\nElenco dei bucket\n')
for bucket in conn.get_all_buckets():
        print('\t{name}\t{created}'.format(
                name = bucket.name,
                created = bucket.creation_date,
        ))

print('\nElenco degli oggetti contenuti nel bucket\n')
for key in bucket.list():
        print ('\t{name}\t{size}\t{modified}'.format(
                name = key.name,
                size = key.size,
                modified = key.last_modified,
                ))

print('\nRecupero dell\'oggetto ciao.txt dal bucket')
key = bucket.get_key('ciao.txt')

print('\nStampa del contenuto dell\'oggetto\n')
print('\t{text}'.format(text = key.get_contents_as_string().decode('utf-8')))

print('\nSalvataggio del contenuto nel file hello.txt')
key.get_contents_to_filename('C:/Users/UTENTE/Documenti/hello.txt')

print('\nRimozione di tutti gli oggetti dal bucket my-new-bucket')
for key in bucket.list(): bucket.delete_key(key.name)

print('\nCancellazione del bucket my-new-bucket')
conn.delete_bucket('my-new-bucket')
```

Una volta eseguito lo script il risultato dovrebbe essere il seguente:

```
Connessione come utente johndoe

Creazione bucket my-new-bucket

Creazione oggetto ciao.txt

Elenco dei bucket

      my-new-bucket   2020-11-19T14:35:51.110Z
```

```
Elenco degli oggetti contenuti nel bucket

      ciao.txt        12      2020-11-19T14:35:51.143Z

Recupero dell'oggetto ciao.txt dal bucket

Stampa del contenuto dell'oggetto

      Hello World!

Salvataggio del contenuto nel file hello.txt

Rimozione di tutti gli oggetti dal bucket my-new-bucket

Cancellazione del bucket my-new-bucket
```

2.6. Requisiti minimi in ambiente di produzione

Ceph è stato progettato per funzionare su hardware di base, il che rende economicamente fattibile la creazione e la manutenzione di cluster di dati su grande scala. Quando si pianifica l'hardware del cluster, è necessario effettuare una serie di considerazioni, inclusi i domini di errore e potenziali problemi di prestazioni. La pianificazione dell'hardware dovrebbe includere la distribuzione di tutti i demoni e di tutti i processi che utilizzano Ceph su molti host; solitamente è consigliata l'esecuzione di un determinato demone su un host che si è pianificato e configurato per ospitare quel tipo di demone. Gli esempi che abbiamo mostrato precedentemente li abbiamo eseguiti utilizzando hardware che non soddisfa i requisiti minimi consigliati in un ambiente di produzione, infatti abbiamo riscontrato parecchi problemi e limitazioni anche se ci è stato comunque possibile realizzare un sistema funzionante sul quale eseguire i test. Nell'immagine in Figura 10 è possibile vedere i requisiti minimi consigliati in ambiente di produzione per ogni demone, inoltre si vuole ricordare che per avere un sistema con funzionalità e proprietà essenziali, è consigliato avere almeno 3 Monitor, 3 OSD e, nel caso si voglia utilizzare CephFS, almeno un MDS.

Process	Criteria	Minimum Recommended
ceph-osd	Processor	• 1 core minimum • 1 core per 200-500 MB/s • 1 core per 1000-3000 IOPS • Results are before replication. • Results may vary with different CPU models and Ceph features. (erasure coding, compression, etc) • ARM processors specifically may require additional cores. • Actual performance depends on many factors including drives, net, and client throughput and latency. Benchmarking is highly recommended.
	RAM	• 4GB+ per daemon (more is better) • 2-4GB often functions (may be slow) • Less than 2GB not recommended
	Volume Storage	1x storage drive per daemon
	DB/WAL	1x SSD partition per daemon (optional)
	Network	1x 1GbE+ NICs (10GbE+ recommended)
ceph-mon	Processor	• 2 cores minimum
	RAM	24GB+ per daemon
	Disk Space	60 GB per daemon
	Network	1x 1GbE+ NICs
ceph-mds	Processor	• 2 cores minimum
	RAM	2GB+ per daemon
	Disk Space	1 MB per daemon
	Network	1x 1GbE+ NICs

Figura 10 - Requisiti minimi per demone

3. Stress test

In questo capitolo andremo a descrivere i test effettuati per verificare alcune caratteristiche e limitazioni della tecnologia.

3.1. Disponibilità degli oggetti a seguito di un fallimento di un OSD

In questa prova si vuole verificare la disponibilità di un oggetto nel caso un OSD dovesse fallire, in particolare se quello in questione è l'OSD identificato come primario. Per farlo ci si pone nella condizione di un cluster interamente funzionante e si effettua una scrittura di un oggetto, in questo caso abbiamo utilizzato lo strumento a linea di comando eseguendo le seguenti istruzioni:

```
ceph osd pool create test-pool
ceph osd pool application enable test-pool test-rados
nano hello.txt

scrivere qualcosa al suo interno e salvare il file

rados -p test-pool put ciao hello.txt
```

Una volta scritto l'oggetto, abbiamo spento un host, ceph-01, cioè quello contenente l'OSD primario sul quale è stata effettuata la scrittura (identificato osservando la

dashboard). A quel punto si è passati ad un host diverso da quello dal quale abbiamo effettuato la scrittura e abbiamo verificato che l'oggetto fosse ancora presente, in seguito lo abbiamo scaricato per verificarne il contenuto:

```
rados -p test-pool ls
rados -p test-pool get ciao hello2.txt
nano hello2.txt
```

In questo modo abbiamo verificato che non sono necessarie tutte le macchine attive per poter leggere e scrivere oggetti, in particolare che non è obbligatoria la presenza dell'OSD primario che ha effettuato la scrittura dell'oggetto nel momento della lettura, fatto dovuto alla configurazione di default del cluster che esegue automaticamente le repliche dei placement group.

3.2. Simulazione blackout

Come seconda prova si è pensato di verificare che il cluster tornasse integro e funzionante a seguito dello spegnimento completo di tutti gli host, simulando un possibile blackout. In questo caso la prova è stata semplice, da una condizione funzionante sono state spente tutte le macchine virtuali e sono state riavviate; una volta che tutti gli host sono tornati online verranno riattivati automaticamente tutti i demoni ed il cluster ritornerà alla condizione iniziale di funzionamento. Inoltre abbiamo notato che nel momento di riaccensione degli host, il sistema si autoconfigura e rigenera automaticamente, senza alcun intervento da parte nostra, che come detto in precedenza è una proprietà chiave di Ceph.

3.3. Scrittura di file di grandi dimensioni tramite CephFS

Come terza prova abbiamo voluto verificare il funzionamento dello storage cluster nel caso di file di grandi dimensioni; per far ciò siamo partiti dalla condizione iniziale di un cluster funzionante composto da 3 OSD, ciascuno con un disco di 15 GB, e abbiamo deciso di utilizzare l'interfaccia CephFS per salvare il file. Abbiamo deciso di verificare tre condizioni al variare della dimensione dei file, la prima con un file molto grande ma che non eccede la dimensione massima di un OSD (un file di circa 10 GB), la seconda con un file che eccede la dimensione di un'OSD ma che è inferiore alla capacità massima di storage del cluster (un file di circa 25 GB mentre la capacità massima del cluster è di 45 GB) e la terza con un file che eccede la capacità massima del cluster. Nella prima situazione volevamo osservare il comportamento nel caso di un file di grande dimensione e abbiamo visto che il sistema non ha avuto problemi in questa operazione, abbiamo inoltre notato che senza cambiare le impostazioni di default viene utilizzata una politica di replicazione (quindi non viene utilizzato l'erasure coding ma deve essere attivato manualmente alla creazione dei pool) a 3 copie quindi in tutto verranno occupati 30 GB della capacità totale (10 GB dell'originale e 20 GB per le due repliche). Nella seconda situazione abbiamo voluto variare leggermente quella precedente, cioè aggiungere un file che superi la capacità massima di ogni singolo OSD ma che rimanga entro i limiti della capacità massima del cluster. Dato che viene effettuato lo striping dei file in oggetti e questi vengono distribuiti in modo uniforme sull'intero cluster, da questa situazione ci si aspetta che non sorga alcun problema, cioè i dati dovrebbero essere distribuiti e dovrebbero occupare in ugual modo tutti gli OSD. Per replicare questa situazione si è pensato di copiare un file di circa 25 GB ma, dato che il cluster ha una capacità massima di circa 45 GB, abbiamo disattivato la replicazione scrivendo una sola copia del file, cioè l'originale; in caso di configurazione di default avremmo occupato 75 GB per le 3

copie, eccedendo la capacità massima dell'intero cluster che verificheremo poi come terza condizione. Il risultato di questa prova è quello previsto anche se si è osservato che la distribuzione dei dati ottenuta è stata un pelo sbilanciata rispetto quella immaginata; comunque anche se non è una distribuzione uniforme esatta è comunque abbastanza equilibrata (OSD1 con 9,4 GB, OSD2 con 9,9 GB e OSD3 con 8,7 GB). L'ultima situazione che abbiamo voluto verificare è il superamento della capacità massima per vedere come si comporta lo storage cluster nel caso venga occupato interamente; per far ciò abbiamo utilizzato il file da 25 GB e abbiamo ripristinato la replicazione. In questa casistica si è pensato a comportamenti totalmente diversi che potevano accadere: poteva essere effettuato un controllo iniziale dal client confrontando lo spazio disponibile con la dimensione del file e delle sue repliche in modo da terminare subito l'operazione con un messaggio di errore; oppure poteva essere effettuata una singola copia del solo file originale ed avvistare l'utente che non è stata effettuata la replicazione a causa di spazio insufficiente; oppure poteva avvenire la copia dando un errore ed annullando il salvataggio una volta raggiunta la capacità massima del cluster. In realtà il comportamento che abbiamo notato è ancora diverso dei precedentemente pensati; il client CephFS non effettua un controllo iniziale, quindi effettua la copia del file nel cluster fino al raggiungimento della capienza massima, momento nel quale il sistema va in panico. Dal client l'operazione non viene interrotta ma rimane in stallo come se la stesse ancora effettuando e non viene segnalato il problema in nessun modo; gli OSD ed il pool utilizzato per il file vengono visualizzati tutti come completamente pieni, inoltre si è notato che gli OSD si sono riavviati tutti ma uno dei tre, ceph-03, è rimasto down. Sono presenti alcune configuarazioni che permettono di attivare dei warning di salute del cluster o addirittura bloccare le scritture dei client, al raggiungimento di determinate percentuali di riempimento dei singoli OSD, in particolare ne esistono tre:

- "Full ratio", di default impostata al 95% della capacità di un OSD, al suo raggiungimento impedisce le operazioni di scrittura da parte dei client;
- "Backfillfull ratio", di default impostato al 90%, indica quanto un nuovo OSD, inserito all'intenro di un determinato active set, può riempirsi tramite le operazioni di ribilanciamento dei dati;
- "Nearfull ratio", definita di default all'85% della capacità degli OSD, al suo raggiungimento genererà un avviso di salute per indicare lo stato dell'OSD in questione, che sarà visibile anche tramite la dashboard.

Durante l'esecuzione della prova queste limitazioni non hanno impedito il riempimento degli OSD e si pensa sia dovuto al fatto che nessun file è stato scritto tra le soglie imposte ed il completo riempimento degli OSD ma è stato utilizzato un file unico che ha riempito completamente tutti gli OSD; in un caso reale probabilmente non potrebbe venirsi a presentare tale condizione in quanto non verrà mai utilizzato Ceph per storage cluster di piccole capacità come nel nostro caso.

3.4. Verifica di situazioni di recupero dai fallimenti

In questa prova abbiamo voluto verificare alcune caratteristiche e condizioni che non abbiamo trovato sulla documentazione, in modo da capire come si comporta lo storage cluster in alcune situazioni. La prima casistica verificata è come vengono gestiti e trattati i manager, abbiamo notato che la gestione degli stati è la stessa descritta precedentemente per gli MDS, cioè questi possono assumere due possibili stati: attivo e standby. Il manager attivo è quello che ospiterà la dashboard e quindi quello al quale dovremo connetterci per ottenere la rappresentazione grafica dello stato del cluster, mentre quello in stato standby sarà in una condizione di riposo, in quanto non effettua operazioni, ma sarà sempre pronto a rimpiazzare il manager attivo qualora

questo fallisse. Al bootstrap del cluster si è notato che vengono inizializzati due manager, uno attivo ed uno standby; abbiamo provato a far fallire quello attivo per verificare la condizione di cui abbiamo appena parlato e in seguito abbiamo fatto fallire anche il secondo manager per verificare cosa fosse successo in questo caso. Nel caso tutti i manager dovessero fallire, Ceph non ne creerebbe di nuovi lasciando lo storage cluster senza tutte le funzionalità offerte da questo componente. Questo vale per tutti i demoni presenti, siano questi essenziali al funzionamento del cluster che non, come ad esempio grafana, prometheus ma anche nel caso questi demoni fossero i monitor; questa casistica è stata verificata in quanto ci si aspettava che per i servizi indispensabili al funzionamento di Ceph fosse riservato un trattamento differente. Nel caso di fallimento di uno dei servizi indispensabili si era previsto che questi sarebbero stati ricreati, qualora fosse disponibile un host configurato per ospitarli ed una condizione che ne richieda la presenza. Per fare tale verifica siamo partiti da un cluster funzionante e abbiamo configurato tutti i monitor per la loro creazione ma ne abbiamo creati solo 3 sugli host ceph-00, ceph-01 e ceph-03 in modo tale da ottenere il quorum; a questo punto abbiamo spento ceph-00, in questo caso il quorum rimane perchè la maggioranza dei monitor è ancora attiva. Una volta spento anche ceph-01, il quorum non è più preservato e di conseguenza non viene più condivisa la cluster map, questo comporta il fatto che i monitor non riusciranno più ad effettuare il loro compito e quindi non sarà più possibile vedere i cambiamenti nello stato del cluter ma solamente l'ultima immagine di esso nell'istante dell'ultimo quorum effettuato. In questa condizione infatti abbiamo visto che spegnendo un manager rimarrebbe comunque invariato il suo stato sulla dashboard. Creando tale situazione ci saremmo aspettati che al primo fallimento di un monitor questo fosse rimpiazzato creandone uno nuovo sull'host ceph-02 ma questo non è successo, e non è stato fatto nemmeno al fallimento del secondo monitor, quando è venuto a mancare il quorum e di conseguenza è stato compromesso il funzionamento dell'intero cluster.

4. Benchmark: Linda

Si è pensato di utilizzare questa tecnologia calandola in un caso reale, in particolare per fornire persistenza agli spazi delle tuple utilizzati dal framework Linda. Per far ciò abbiamo deciso di utilizzare la funzionalità RadosGW vista in precedenza, in quanto è quello che si presta meglio a rappresentare il modello utilizzato da Linda, in particolare è possibile raffigurare gli spazi di tuple come bucket S3 e le tuple come gli oggetti al loro interno. Come prima cosa abbiamo completato l'esercizio del laboratorio 7, implementando le funzionalità necessarie per il funzionamento del client Linda, in seguito lo abbiamo modificato per aggiungere la persistenza dello spazio delle tuple tramite l'utilizzo di Ceph. Si è pensato di mantenere il funzionamento del client invariato sfruttando la memoria locale per le operazioni in sola lettura, ed interagire con lo storage cluster solamente per la creazione dello spazio delle tuple e per le operazioni "in" ed "out". Inoltre, è stato necessario modificare le tuple, in quanto ognuna di esse dovrà ora possedere anche un riferimento al nome dell'oggetto con cui sarà memorizzata su Ceph. Dato che lo spazio delle tuple dovrà essere persistente, sarà necessario controllare se questo esiste già all'interno di Ceph, in caso contrario si crea un nuovo bucket, altrimenti si ottiene il riferimento di quello già presente, si leggono tutte le tuple al suo interno e si memorizzano nello spazio delle tuple locale. Le operazioni di "in" ed "out" sono state modificate in modo da rispecchiare le operazioni locali sulla memoria persistente, rimuovendo gli oggetti corrispondenti alle tuple rimosse dalle "in" ed aggiungendoli in caso di "out"; per questa seconda operazione si è pensato inoltre di non scrivere in memoria persistente le tuple al momento della ricezione della richiesta, ma

solamente dopo aver completato le richieste pendenti, il cui template viene soddisfatto dalle tuple in questione, ed alla condizione che queste tuple non vengano immediatamente rimosse da tali richieste. In questo modo si possono evitare operazioni di scrittura di tuple dalla memoria persistente, in quanto verrebero subito rimosse dalle richieste pendenti che ne soddifano il match, riducendo così il tempo di esecuzione delle operazioni "out". Per verificare che il comportamento del client sia quello atteso abbiamo aggiunto un test, per verificare la persistenza dei dati, a quelli già presenti che utilizziamo per assicurarci che il comportamento rimanga consistente; questo test compie i seguenti passi:

1. crea uno spazio delle tuple e vi aggiunge 4 tuple;
2. crea un nuovo spazio delle tuple con lo stesso nome, dato che sarà già presente caricherà le tuple create nel passo precedente;
3. richiede il contenuto dello spazio delle tuple creato nel passo 2;
4. perchè il test abbia successo, verificare che il risultato del punto 3 e l'insieme iniziale delle tuple coincidano.

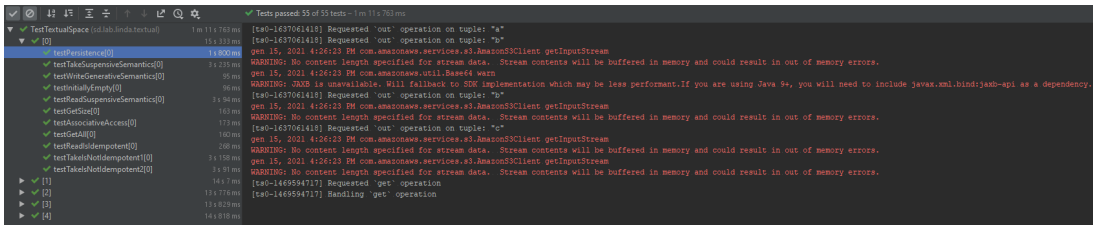

Figura 11 - Risultato del test

5. Pros & Cons e scenari di utilizzo

Ceph è una tecnolgia ben consolidata per la realizzazione di sistemi di storage distribuiti, in quanto i vantaggi che offre sono rappresentati da feature molto ricercate sia per semplici soluzioni, sia per grandi ambienti di produzione. In particolare Ceph:

- offre ottime prestazioni in termini di velocità delle operazioni, e risulta essere una soluzione molto scalabile;
- presenta ottime integrazioni con le principali tecnologie di virtualizzazione (come KVM, libvirt, OpenStack, XEN e Proxmox). Questo unito alla possibilità di immagazzinare snapshot all'interno del proprio sistema di storage rende Ceph un'ottima scelta per supportare la virtualizzazione;
- offre tecnologie per operare con tutte e tre le tipologie di storage quali blocchi, file e immagini, rendendolo dunque estremamente flessibile ai vari scenari applicativi (come ad esempio NAS e SAN);
- è una tecnologia open source, ideale per le aziende che vogliono ridurre il costo dei propri sistemi di storage, senza però sacrificarne l'efficacia;
- fa in modo che tutti i dati siano replicati automaticamente da un nodo ad un altro, evitando il disservizio qualora uno dei nodi dovesse fallire, dato che potrebbe essere fornito dal nodo ancora attivo. In questo modo i dati possono essere recuperati in un qualsiasi momento;
- immagazzina in modo efficiente i dati, assicurandosi di non occupare spazio in memoria finchè non sono effettivamente stati salvati;

- è altamente resiliente, grazie alla sua capacità di rilevare e riprendersi automaticamente da eventuali fallimenti, fintanto che viene mantenuto il numero minimo di nodi attivi.

Ad ogni modo Ceph presenta ancora alcune problematiche e svantaggi, riscontrati dai molti utenti che ne hanno fatto uso, in particolare:

- la realizzazione di un cluster Ceph completo risulta essere un processo estremamente lungo e complesso. In questo viene in limitato aiuto la documentazione, che in molti casi risulta essere estremamente frammentaria e talvolte obsoleta;
- il sistema di bootstrap del cluster risulta minimo nei report degli errori, per cui gli utenti faticano a comprendere se i processi di setup sono andati a buon fine;
- errori nella configurazione possono risultare in cali di performance e distribuzione dei dati sbilanciata;
- per ambienti di produzione è sconsigliata la realizzazione di un cluster Ceph risiedente su macchine virtuali per motivi di ottimizzazione. Questo può essere un fattore limitante per piccoli casi d'uso che non possono permettersi l'utilizzo di decine di macchine fisiche, ognuna con il proprio OSD.

Scenari applicativi

Ceph può essere utilizzato in molti scenari applicativi, uno dei principali è il suo affiancamento alle principali tecnologie per la realizzazione di database Sql e NoSql. In particolare Ceph è stato adottato da molte aziende per la realizzazione dei propri servizi di Database-as-a-Service (DaaS), sfruttando le proprie risorse hardware invece di ricorrere al public cloud. Gli storage cluster Ceph possono gestire diversi tipi di carichi di lavoro; sebbene quelli ad alto rendimento ed incentrati sul costo/capacità siano comuni, l'utilizzo di Ceph per servire carichi di lavoro MySQL ad alta intensità di operazioni input/output al secondo (IOPS) è sempre più visto negli ambienti di produzione per la sua ottima propensione alla scalabilità. Ceph consente a gruppi di server di mettere in comune le proprie risorse di archiviazione e permette a qualsiasi server del gruppo di accedere all'archiviazione sulla rete. Ciò consente un posizionamento più flessibile ed efficiente dei database nel cluster, senza richiedere strumenti come rsync o la replica di MySQL per spostare i dati. Inoltre offre un'ampia gamma di funzionalità, tra cui: ridimensionamento live dei volumi, mirroring dei volumi, migrazione dei volumi, migrazione live delle macchine virtuali, clonazione copy-on-write e snapshot. Ceph è quindi un'ottima soluzione per la gestione di grandi raccolte di istanze di database MySQL attraverso l'utilizzo dell'interfaccia RBD. Le aziende che hanno deciso di adottare questo modello di cloud privato o ibrido basato su Ceph, sono in grado di fornire dei sistemi di storage ad un prezzo migliore del già popolare cloud pubblico, senza sacrificare caratteristiche di performance essenziali.

Un'altro possibile scenario applicativo è l'utilizzo di Ceph per la realizzazione di Network Attached Storage (NAS) clusterizzati. Una web server farm deve avere accesso agli stessi file in modo che i client connessi ricevano lo stesso contenuto a prescindere dal server a cui si connettono. Tradizionalmente per offrire accesso ai file distribuiti viene utilizzato un NAS ad alta disponibilità, ma questo può riscontrare diverse limitazioni quando si tratta di scalare il sistema. In questo contesto si fa strada CephFS, in quanto grazie alla sua capacità di realizzare sistemi di storage basati su file, è in grado di distribuire dati e metadati sui vari nodi del cluster, fornendo così accesso unificato ai file da ciascun nodo a prescindere dall'effettiva locazione dei dati. Questo gli permette di essere adottato per

immagazzinare i contenuti su un file system distribuito ed essere montato su ogni nodo della server farm, ponendosi come alternativa ai NAS precedentemente adottati.

Ceph inoltre sta diventando un'alternativa molto allettante per la realizzazione di Storage Area Network (SAN), poichè in grado di fornire le tre tipologie principali di storage (file, blocchi e oggetti) sfruttando hardware relativamente economico rispetto alle metodologie stabilite. In questo modo permette alle aziende di sfuggire al vendor lock-in senza compromettere però le performance e senza mancare di feature importanti. Una delle integrazioni di RBD è la possibilità di utilizzare un gateway iSCSI; questo presenta un'entrypoint iSCSI ad alta disponibilità che esporta le immagini RBD (RADOS Block Device) come dischi SCSI. Il protocollo iSCSI consente ai client di inviare comandi SCSI ai dispositivi di archiviazione su una rete TCP/IP, consentendo a quelli senza il supporto client Ceph nativo di accedere allo storage a blocchi Ceph. Con questa funzionalità è possibile fornire un'infrastruttura di storage a blocchi completamente integrata con tutte le caratteristiche ed i vantaggi di una rete SAN convenzionale.

Infine Ceph può essere utilizzato da alcuni dei principali hypervisor (VMware, Xen, Proxmox, QEMU, KVM etc..) per gestire lo storage di immagini e snapshot di macchine virtuali, permettendo inoltre la gestione dei dischi virtuali delle singole macchine sfruttando l'interfaccia RBD, la quale effettuerà lo striping dei volumi suddividendoli in blocchi e distribuendoli nel cluster. Dato che le immagini non vengono immagazzinate sullo stesso hardware su cui risiede l'hypervisor, diventa possibile effettuare live migration delle macchine, cioè spostare le macchine virtuali tra macchine guest senza doverle spegnere e senza disconnettere i client, trasferendo automaticamente memoria, storage e connessioni della macchina virtuale.

Bibliografia

- Sage A. Weil, Andrew W. Leung, Scott A. Brandt, Carlos Maltzahn. RADOS: A scalable, reliable storage service for petabyte-scale storage clusters, 2007
- Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Carlos Maltzahn. CRUSH: Controlled, Scalable, Decentralized Placement of Replicated Data, 2006.
- Singh Karan, Weil Sage. Learning Ceph. Birmingham, England: Packt Publishing, 2015
- Vikhyat Umrao, Michael Hackett, Karan Singh. Ceph Cookbook - Second Edition: Practical recipes to design, implement, operate, and manage Ceph storage systems. Birmingham, England: Packt Publishing, 2017
- Sage A. Weil. Ceph: reliable, scalable, and high-performance distributed storage, 2007
- Documentazione Ceph di Red Hat, https://access.redhat.com/documentation/en-us/red_hat_ceph_storage/4/html/architecture_guide/the-ceph-architecture_arch (data di consultazione: dicembre 2020)
- Documentazione ufficiale di Ceph, https://docs.ceph.com/en/latest/ (data di consultazione: dicembre 2020)
- Video di Ross Turk (Inktank). Ceph intro & architectural overview, https://www.youtube.com/watch?v=7I9uxoEhUdY
- Video di Sage Wail. Intro to Ceph, https://www.youtube.com/watch?v=PmLPbrf-x9g
- Articolo Medium di fajlinuxblog. Ceph: an overview, https://fajlinuxblog.medium.com/ceph-an-overview-e971c00ded93