

SR05 - Étude CEPH

FAN YIZHEN
FERRY CLÉMENT
LIBRAIRE AUDRICK
LIU YAN
NIE YANYAN
WANG JIAYANG
XIAO WEIXUAN

Contents

| | | |
|----------|---|-----------|
| 1 | Systèmes de stockage distribués général | 2 |
| 1.1 | Introduction | 2 |
| 1.2 | Pourquoi choisir une solution de stockage distribué ? | 2 |
| 1.3 | Diverses solutions de système de stockage distribué | 2 |
| 1.4 | Leurs avantages, inconvénients et Déploiement | 3 |
| 1.4.1 | AFS | 3 |
| 1.4.2 | HDFS(Hadoop) | 3 |
| 1.4.3 | Gluster | 4 |
| 1.4.4 | Lustre | 5 |
| 1.4.5 | MooseFS | 5 |
| 2 | Compréhension de l'architecture de Ceph | 6 |
| 2.1 | Introduction | 6 |
| 2.1.1 | OSD (Object Storage Daemon) | 7 |
| 2.1.2 | MON (Monitor) | 7 |
| 2.1.3 | MGR (Manager) | 7 |
| 2.1.4 | MDS (MetaData Server) | 7 |
| 2.1.5 | Pool et PGs (Placement Groups) | 8 |
| 2.1.6 | RBD (RADOS Block Devices) | 8 |
| 2.1.7 | RGW (Rados gateway) | 8 |
| 2.1.8 | CephFS (Ceph FileSystem) | 9 |
| 2.2 | Hash Uniforme | 10 |
| 2.2.1 | La tolérance de faute et l'extensibilité de Hachage | 10 |
| 2.2.2 | Problème d'asymétrie des données dans l'anneau de hachage | 11 |
| 2.2.3 | Hachage dans Ceph | 12 |
| 2.3 | Algorithme CRUSH | 13 |
| 2.3.1 | Cluster Map | 14 |
| 2.3.2 | Le processus | 15 |
| 2.3.3 | Fonction SELECT | 16 |
| 2.4 | Algorithme PAXOS | 16 |
| 2.4.1 | Algorithme PAXOS original | 17 |
| 2.4.2 | PAXOS dans CEPH | 18 |
| 2.4.3 | Algorithme de CEPH PAXOS | 18 |
| 2.4.4 | Sélection consensus de CEPH PAXOS | 21 |
| 2.5 | L'intérêt du système, concurrence, avantage du Ceph par rapport aux autres systèmes | 22 |
| 3 | Évaluation des performances (cas général de Ceph) | 23 |
| 3.1 | Les métriques pertinentes et les paramètres à faire varier | 23 |
| 3.2 | Étude de notre cas (Snapshot des résultats dans l'annexe A) | 23 |
| 4 | Réalisation | 25 |
| 5 | Bibliographie | 29 |
| A | Benchmark du Ceph | 29 |

1 Systèmes de stockage distribués général

1.1 Introduction

Dans les années 2000/2010, les entreprises préféraient les solutions de stockage centralisées, elles sont de plus en plus séduites par les solutions distribuées. Ceph est notamment utilisé par des entreprises spécialisées dans la finance telles que Bloomberg et Fidelity. Par conséquent, les ingénieurs en informatique doivent être capable de faire face à cette transition en connaissant les différentes solutions existantes pour choisir la plus adaptée (Occupation du réseau, etc...).

C'est dans l'optique de pouvoir faire face à cette problématique dans notre vie professionnelle que nous avons décidé de nous lancer dans ce projet. L'étude détaillée d'une solution distribuée permet d'acquérir une meilleure appréhension du domaine.

1.2 Pourquoi choisir une solution de stockage distribué ?

"Distribué" est un type d'architecture système constitué d'un ensemble de noeuds d'ordinateurs qui communiquent via un réseau et se coordonnent pour travailler ensemble afin d'accomplir des tâches communes. L'apparition des systèmes distribués a permis d'effectuer des tâches de calcul et de stockage qui ne peuvent pas être effectuées par un seul ordinateur. Un des objectifs du stockage distribué est d'utiliser plus de machines pour stocker plus de données.

Les systèmes distribués se développent en se basant sur les systèmes centralisés. Mais le concept des deux systèmes est complètement opposé. Un système centralisé est une architecture qui distribue tous les programmes et fonctions à une machine unique pour fournir des services. Elle est facile à maintenir, cependant les inconvénients sont également évidents. Si la machine centrale tombe en panne, le système entier cesse de fonctionner.

Éviter la vulnérabilité d'un seul noeud dans le système centralisé n'est qu'une des raisons pour utiliser le système distribué. Une autre raison importante est l'extensibilité. Après tout, toutes les machines (même un super-ordinateur) auront des limites de performance. Cependant, les systèmes distribués peuvent être améliorés horizontalement en augmentant le nombre de machines.

Avec l'avènement de l'ère Internet, la quantité de données à stocker a augmenté de façon importante. Traiter des demandes simultanées élevées et des données volumineuses devient un problème urgent en entreprise. Dans le même temps, les exigences des applications d'entreprise changent souvent avec le temps, ce qui impose également des exigences élevées aux plates-formes d'applications d'entreprise. La flexibilité des services proposés par le système distribué est donc l'avantage principal pour l'adaptation des entreprises aux changements des demandes. Celui-ci est aussi le moteur du développement du système distribué.

De plus, les solutions de stockage distribués sont également l'opportunité d'avoir une gestion de sauvegarde des fichiers d'un noeud sur ses noeuds voisins.

1.3 Diverses solutions de système de stockage distribué

Aujourd'hui, il existe plusieurs solutions pour le système de stockage distribué dans le monde entier.

- AFS (Andrew File System) qui est inspiré par NFS (Network File System). Les différentes unités de stockage du réseau sont montées dans un répertoire commun. C'est le premier système de stockage distribué réalisé par CMU et IBM en 1982.
- HDFS (Hadoop Distributed File System) qui est inspiré par la publication de MapReduce, GoogleFS et BigTable de Google.
- GlusterFS est un système de fichiers libre distribué en parallèle, qui permet de stocker jusqu'à plusieurs péta-octets (10^{15} octets). C'est un système de fichiers de grappe de serveurs.
- Lustre est un système de fichiers distribué libre, généralement utilisé pour de très grandes grappes de serveurs. Le nom provient de la fusion entre les mots Linux et Cluster. Il est maintenant développé, distribué et maintenu par Sun Microsystems.
- Swift est un service sous openstack qui permet de créer et gérer des espaces de stockage objet en ligne. Ce service est stable et sûr d'utilisation. Il permet de stocker des fichiers de très grande capacité et accessibles de manière instantanée. Swift permet le stockage objet persistant, sécurisé, redondant, connecté directement à internet sans instance de serveur, sans contrainte de réservation d'espace.

- Moose est un système de fichiers distribué développé par Gemius SA. MooseFS se veut tolérant aux pannes, redimensionnable, compatible POSIX, et générique. Il suit en grande partie les mêmes concepts que Google File System, Lustre ou Ceph. Le code préalablement propriétaire a été libéré et mis à disposition publiquement le 5 mai 2008.

1.4 Leurs avantages, inconvénients et Déploiement

1.4.1 AFS

Avantages :

- Meilleure performance réseau : AFS a été conçu pour les grands réseaux, donc plus rapide et plus efficace.
- Authentification d'utilisateur : AFS authentifie les utilisateurs, pas les machines. Cela permet aux personnes ayant un compte AFS d'accéder à leurs fichiers de compte à partir de n'importe quel ordinateur équipé du système de classement AFS, tel que ceux du cluster Terman.
- Système de fichiers global : chaque institution utilisant AFS et connectée à Internet possède probablement une cellule.
- Administration simplifiée : les administrateurs peuvent se concentrer sur les serveurs plutôt que sur les ordinateurs clients, car ces derniers obtiennent toutes leurs informations.
- Autorisations de fichiers : AFS permet un meilleur contrôle des autorisations de fichiers de ses répertoires.
- Sauvegardes : chaque nuit, le serveur AFS crée une sauvegarde de votre compte à laquelle vous pouvez accéder.
- Sécurité : AFS, associé à Kerberos, offre une excellente sécurité car il authentifie les utilisateurs et les tâches, et non les machines (cela réduit les risques d'usurpation d'adresse IP).

Inconvénients :

- AFS n'utilise pas la sémantique de fichier Unix : il ne supporte que les permissions de répertoire. Cela peut semer la confusion chez les utilisateurs habitués aux ensembles d'autorisations standard Unix `chmod`.
- L'authentification peut poser problème : le besoin de tout authentifier et tout le monde peut parfois poser problème à ceux qui n'ont aucune idée de la façon de gérer l'authentification.
- Certains fichiers spéciaux utilisés dans Unix standard ne sont pas pris en charge par AFS.

1.4.2 HDFS(Hadoop)

HDFS est utilisé par : Yahoo!, Microsoft, LinkedIn, Facebook, ...

HDFS est basé sur une architecture Client/Serveur. Le "NameNode" est le serveur maître qui gère le système de fichiers et régule les accès aux fichiers pour les clients.

Avantages :

- Il est distribué sur des centaines voire des milliers de serveurs, et chaque nœud stocke une partie du système fichier. Pour éviter le risque de perdre des données, chaque donnée est stockée à trois emplacements. Très efficace pour le traitement de flux de données.
- Il est très adapté au Big Data. Des dizaines de millions de fichiers peuvent être supportés sur une seule instance. Par ailleurs, cela assure la cohérence et l'intégrité des données pour éviter tout désagrément.
- HDFS évite la congestion de réseau en privilégiant le déplacement des opérations plutôt que le déplacement des données. Ainsi, les applications peuvent accéder aux données à l'endroit où elles sont entreposées.
- Il peut fonctionner sur différents types de commodity hardware sans aucun problème de compatibilité.

Inconvénients :

- N'autorise pas l'accès aux données à faible temps de latence, telles que les millisecondes, pour stocker des données. Par exemple, lire des données en quelques millisecondes est donc très difficile à faire.
- Problèmes liés aux petites données, il ne peut pas fonctionner efficacement dans de petits environnements de données.
- Écriture simultanée, modification aléatoire d'un fichier ne peut avoir qu'une seule écriture, il n'est pas autorisé à écrire plusieurs threads en même temps. Seul l'ajout de données est pris en charge, et la modification aléatoire de fichiers n'est pas prise en charge.

Déploiement :

- Voyage en ligne : Actuellement, 80% des sites Web de voyages en ligne dans le monde utilisent la distribution Hadoop fournie par Cloudera.
- Données mobiles : Hadoop prend en charge 70% des services de données pour smartphones aux États-Unis.
- E-commerce : eBay
- Energy mining : Chevron est la deuxième plus grande société pétrolière aux États-Unis. Elle utilise Hadoop pour collecter et traiter des données afin de pouvoir localiser la mine.
- Traitement des images : la startup Skybox Imaging utilise Hadoop pour stocker et traiter les données d'image, ainsi que pour détecter les changements géographiques liés aux images haute définition capturées par satellite.
- Sécurité informatique : outre la gestion de l'infrastructure informatique de l'entreprise, Hadoop peut également être utilisé pour traiter des données générées par une machine afin d'identifier les attaques de logiciels malveillants ou de réseaux.

1.4.3 Gluster

GlusterFS est maintenant une acquisition : Red Hat.

GlusterFS est dockerisé (image docker disponible pour créer des noeuds), GlusterFS supporte la virtualisation des datastores de KVM et de RHEV.

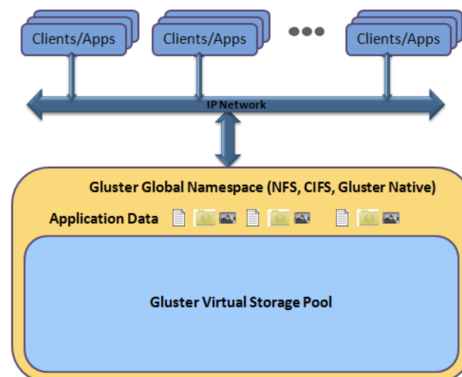


Figure 1: Gluster

Avantages :

- Gluster ne nécessite aucun serveur de méta-données afin de savoir où les données se situent sur le cluster.
- Stockage flexible : les clusters Gluster peuvent augmenter ou réduire de manière flexible le stockage de données et ajouter ou supprimer des ressources dans un pool de stockage en fonction des besoins de l'entreprise sans perturber le fonctionnement du système.

- Évolutivité : les clusters de palettes peuvent augmenter la capacité ou les performances de l'ensemble du système en ajoutant des nœuds de stockage.
- Fiabilité élevée : le cluster Gluster garantit la fiabilité des données grâce à la réplication et à la réparation automatique (à l'aide de journaux du système de fichiers sur disque tels que EXT3 ou ZFS)

Inconvénients :

- Performance des méta-données : GlusterFS utilise un algorithme de hachage flexible pour remplacer les services de méta-données centralisées ou distribuées dans les systèmes de fichiers distribués traditionnels, ce qui le rend très rapide pour rechercher et localiser un nom de fichier donné. Toutefois, si vous ne connaissez pas le nom de fichier au préalable, les performances chuteront considérablement si vous répertoriez le répertoire de fichiers (ls ou ls -l).
- Problème lié aux petits fichiers : analyse théorique et pratique, GlusterFS est actuellement principalement utilisé pour les scénarios de stockage de fichiers volumineux, pour les petits fichiers, en particulier les fichiers volumineux, les performances de stockage et les performances d'accès ne sont pas satisfaisantes.
- Équilibrage de la capacité : GlusterFS doit effectuer manuellement l'équilibrage de la charge sur les nœuds ajoutés ou supprimés, sans tenir compte de la charge système actuelle et pouvant affecter l'accès normal au service. L'équilibrage de la charge de la capacité GlusterFS est réalisé en montant des volumes sur le nœud d'exécution actuel, puis en copiant, supprimant et renommant des fichiers. Il n'est pas exécuté simultanément sur tous les nœuds du cluster et les performances de l'équilibrage de la charge sont médiocres.
- Problèmes de disponibilité des données : les volumes de réplication GlusterFS sont mis en miroir par fragments. Ce mode n'est pas très flexible. La relation de réplication de fichiers ne peut pas être ajustée de manière dynamique. Dans l'éventualité d'une réplique défaillante, la disponibilité du système sera quelque peu réduite.

Déploiement GlusterFS est utilisé par : Amazon, Red Hat, etc...

1.4.4 Lustre

Déploiement

Lustre est utilisé par la plupart des superordinateurs TOP500 et des grands sites multi-grappes. Six des 10 et plus de 60 des 100 supercalculateurs utilisent les systèmes de fichiers Lustre.

1.4.5 MooseFS

Avantages :

- Le déploiement et l'installation sont très simples et faciles à gérer.
- Prendre en charge le mécanisme d'extension en ligne pour améliorer l'extensibilité du système.
- Mise en œuvre d'un RAID logiciel, d'une capacité améliorée de traitement simultané du système et d'une capacité de récupération de la tolérance de panne des données.
- La récupération de données est plus facile et améliore la disponibilité du système.
- Il existe une fonction de corbeille, personnalisation des affaires.

Inconvénients :

- Le nœud MFS master consomme de la mémoire.
- Pour les fichiers de moins de 64 Ko, l'utilisation du stockage est faible.

Déploiement

Application de déploiement de cluster unique, fichiers de taille moyenne et grande, petites et moyennes entreprises, écoles, sites Web.

Tableau comparatif fonctionnel

| HDFS VS CEPH VS GLUSTERFS – FONCTIONNALITÉS | HDFS | CEPH (0.70) | GLUSTERFS (3.4) |
|---|-------------------------|--------------------|-----------------|
| Architecture | Centralisée | Distribuée | Décentralisée |
| Gestion des Nom | Index | CRUSH | EHA |
| API | CLI, FUSE, REST, API | FUSE, mount, REST | FUSE, mount |
| Détection des pannes | Connexion Complète | Connexion Complète | Détectée |
| Disponibilité Système | Pas de Failover (natif) | Haute | Haute |
| Disponibilité de la donnée | Réplication | Réplication | Comme du RAID |
| Stratégie de répartition | Automatique | Automatique | Manuel |
| Réplication | Asynchrone | Synchrone | Synchrone |
| Consistance du Cache | WORM, lease | Lock | ✗ |
| Load Balancing | Automatique | Manuel | Manuel |

Figure 2: Tableau comparatif de différentes technologies

2 Compréhension de l'architecture de Ceph

Ceph est une plateforme libre de stockage distribué, dont les objectifs principaux sont d'être complètement distribué sans point unique de défaillance, extensible et librement disponible. Les données stockées dans le système de Ceph sont répliquées, ce qui permet la tolérance aux pannes des systèmes.

2.1 Introduction

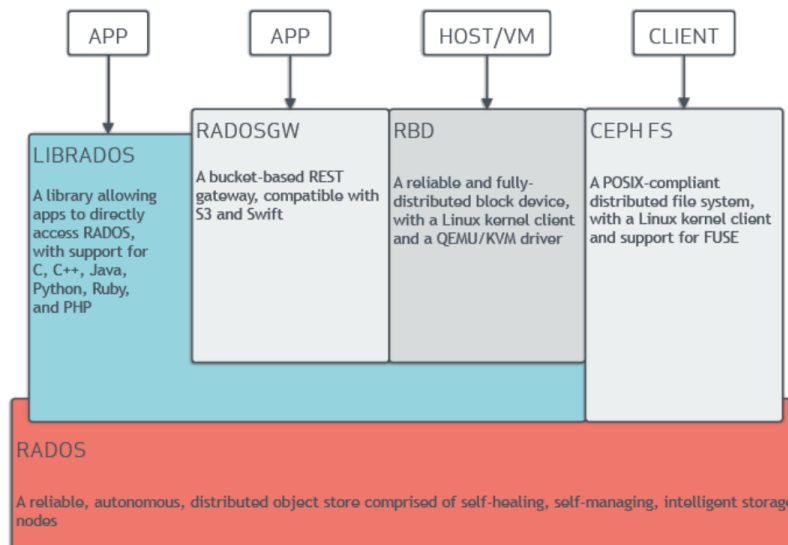


Figure 3: L'architecture de Ceph

La couche sous-jacente de Ceph est RADOS (Reliable Autonomic Distributed Object Store), qui est aussi un système de stockage distribué. RADOS expose l'interface d'appel LIBRADOS au monde extérieur, il ne

suffit que d'appeler l'interface LIBRADOS pour manipuler Ceph. Les fonctions de stockage de Ceph sont basées sur RADOS. RADOSGM (Rados Rest Gateway) est utilisé pour le stockage d'objets; RBD (Rados Block Devices) pour le stockage de blocs; CephFS (Ceph FileSystem) est un programme en mode noyau qui fournit une interface POSIX au monde extérieur.

L'unité de base de fonctionnement de Ceph est un noeud. Un noeud de Ceph exploite le matériel standard et les démons intelligents. Un grand nombre de noeuds fournit un cluster de stockage de Ceph. Les noeuds communiquent entre eux pour répliquer et redistribuer les données de manière dynamique. Un cluster de stockage de Ceph contient quatre types de noeuds: Ceph monitor, Ceph OSD Daemon, Ceph Manager et Ceph MetaDate Server.

Dans le système distribué de stockage, comment on applique des règles de la distribution des données et de l'adressage des données est toujours un sujet de préoccupation. Normalement, tous les rôles (serveurs, clients) dans le système respectent un algorithme unitaire pour réaliser ses fonctions.

Tout d'abord, la résolution est de maintenir une table (key-value) globale. Il suffit que les rôles consultent la table globale lors de la manipulation des données. Cependant, avec l'augmentation de la quantité des données et l'élargissement de la taille de cluster, il devient de plus en plus difficile de maintenir une table globale. L'algorithme CRUSH est destiné de résoudre cette problématique. CRUSH n'a pas besoin que des informations et des règles prédéfinies qui décrivent l'architecture physique de cluster (conclus dans CRUSH map) pour réaliser la fonction de l'adressage des données.

Quant à la distribution des données dans le système distribué, il faut généralement considérer les 3 facteurs ci-dessous :

1. L'isolation des domaines de défaillance. Les répliqués d'une même données doivent être distribués dans des différents domaines de défaillance pour réduire le risque de la détérioration des données.
2. La répartition de charge. C'est à dire que des données sont distribués uniformément parmi des noeuds de stockage pour éviter le surcharge ou l'oisiveté des noeuds, qui affectera la performance du système.
3. Le contrôle de la quantité de la migration des données lors de la modification (l'ajout ou la suppression) d'un nouveau noeud. Le meilleur cas est que lors de la modification d'un noeud, seul les données situées dans ce noeud vont être migrées vers les autres noeuds, et des données sur les noeuds normaux ne vont pas faire la migration.

2.1.1 OSD (Object Storage Daemon)

Les OSDs servent au stockage de toutes les données et objets du cluster. Ils permettent de dater la répliquion, la récupération, le renvoi et le rééquilibrage des données du cluster. Un OSD peut envoyer des pulsations aux autres démons OSD, et fournir des informations de surveillance à monitor.

Au lieu d'une table de consultation centrale, les client du cluster de stockage et chaque Ceph OSD utilisent l'algorithme CRUSH pour calculer efficacement les informations sur l'emplacement des données.

2.1.2 MON (Monitor)

Un Ceph monitor conserve une master copie du cluster Map. Le Cluster Map décrit les OSDs inclus dans le cluster de stockage et la distribution de toutes les données. Un cluster de Ceph monitors garantit une haute disponibilité en cas d'échec d'un démon de monitor. Les clients du cluster de stockage récupèrent une copie de la carte du cluster de Ceph Monitor. Autrement dit, le client se connecte d'abord à Monitor par TCP, obtient le Cluster Map et effectue des calculs sur le client. Une fois l'emplacement de l'objet connu, il communiquera directement avec l'OSD (décentralisée).

2.1.3 MGR (Manager)

Le Ceph Manager travaille avec le Ceph Monitor pour fournir une surveillance ainsi que des interfaces supplémentaires aux systèmes de surveillance et de gestion externes. Après la version 12.x(luminous), Un Ceph Manager est obligatoire pour le système.

2.1.4 MDS (MetaData Server)

Le Ceph MetaDate Server n'est généralement pas nécessaire, car il n'est pas requis qu'on utilise CephFS. Il peut maintenir les informations de fichiers dans le CephFS et les fournit aux CephFS clients. Avec ce serveur,

pour les commandes qui ne manipule pas les données de fichiers dans CephFS, par exemple `ls`, il n'y aura pas de coût pour le cluster. Toutes les informations nécessaires sont déjà stockées dans MDS.

2.1.5 Pool et PGs (Placement Groups)

Un cluster de stockage de Ceph peut être divisé en plusieurs pools. Chaque pool isole entre eux mutuellement et logiquement. Les différents pools peuvent avoir des méthodes de traitement des données complètement différentes, telles que le nombre de PGs, la taille de réplique et les règles CRUSH.

Dans Ceph, PG (Placement Group) est une notion logique. Elle est similaire à l'index de la base de données lorsque les données sont adressées : chaque objet Ceph est associé de manière fixe dans une PG. Lors de la recherche d'un objet, il suffit de rechercher la PG à laquelle appartient l'objet, puis la parcourir. Dans ce cas, il n'est pas nécessaire de parcourir tous les objets. De plus, PG est également utilisé comme unité de base pour la migration.

Comme nous avons dit, la PG permet d'organiser et cartographier le stockage d'objet. En effet, une PG est responsable de l'organisation de plusieurs objets (milliers, voire davantage), mais un objet ne peut être associé qu'à une seule PG. La correspondance entre la PG et l'objet est "un à plusieurs". En plus, une PG sera associée sur N OSDs et chaque OSD comportera un grand nombre de PG. C'est à dire la correspondance "plusieurs à plusieurs" entre la PG et l'OSD.

2.1.6 RBD (RADOS Block Devices)

Dans les systèmes Unix ou Linux, il y a deux types de périphériques : le périphérique en caractère et le périphérique en bloc. Un périphérique en caractère se présente souvent comme un appareil d'entrée ou de sortie. Un bloc est une séquence d'octets (par exemple, un bloc de données de 512 octets). Il peut se présenter en forme d'image. Les interfaces de stockage basées sur des blocs constituent le moyen le plus courant de stocker des données sur des supports en rotation tels que des disques durs, des CD, des disquettes et même des bandes traditionnelles à 9 pistes. L'omniprésence des interfaces de périphériques en blocs fait d'un périphérique en bloc virtuel un candidat idéal pour interagir avec un système de stockage de données de masse comme Ceph.

Ceph RBD interfaces utilisent aussi le système de stockage d'objet (OSD) qui fournit l'interface de librados et le système de fichier CephFS. Ceph RBD sauvegarde les images de bloc comme objets dans OSD. Grâce à librados, RBD hérite les capacités de librados, par exemple, le snapshot, la récupération de snapshot.

En plus, Ceph peut diviser des images pour plusieurs nœuds dans un cluster. Cela va beaucoup améliorer la performance d'accès de l'image large.

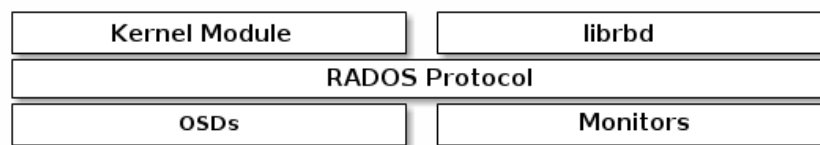


Figure 4: RADOS Block Devices (RBD)

2.1.7 RGW (Rados gateway)

Rados Gateway est une interface de stockage d'objets construite sur librados pour fournir aux applications avec une RESTful gateway vers les clusters Rados. Il prend en charge deux interfaces:

1. S3-compatible : Fournit une fonctionnalité de stockage en mode bloc avec une interface compatible avec un grand sous-ensemble de Amazon S3 RESTful API.
2. Swift-compatible: Fournit une fonctionnalité de stockage en mode bloc avec une interface compatible avec un grand sous-ensemble de OpenStack Swift API.

Rados Gateway est un module FastCGI permettant d'interagir avec librados. Puisqu'il fournit des interfaces compatibles avec OpenStack Swift et Amazon S3, Rados Gateway dispose de sa propre gestion

d'utilisateurs. Rados Gateway peut stocker des données dans le même cluster Rados que celui utilisé pour stocker les données de CephFS client ou Rados Block Devices. Les API S3 et Swift partagent un espace de noms commun. Nous pouvons donc écrire des données avec une API et les récupérer avec l'autre. Rados Gateway n'utilise pas CephFS metadata server.



Figure 5: RADOS Gateway

2.1.8 CephFS (Ceph FileSystem)

CephFS est un système de fichiers distribué conforme à POSIX et construit sur la grappe de serveurs (cluster) Ceph. Il utilise le même Ceph Storage Cluster système que Ceph Block Devices, Ceph Object Gateway ou librados API.

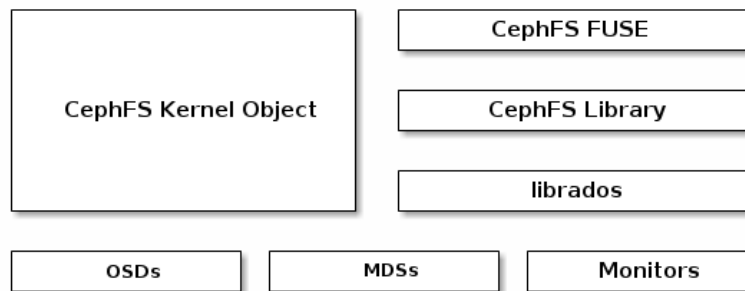


Figure 6: Ceph FileSystem

Pour exécuter le système de fichiers Ceph, nous devons avoir un Ceph Storage Cluster en cours d'exécution avec au moins un Ceph Metadata Server (MDS).

CephFS introduit les avantages suivants :

1. Évolutivité

CephFS est hautement évolutif car les clients lisent et écrivent directement sur tous les noeuds OSD.

2. Système de fichiers partagés

CephFS est un système de fichier partagé permettant à plusieurs clients de travailler simultanément sur le même système de fichiers.

3. La haute disponibilité

CephFS fournit un cluster de MDS. L'un est actif et les autres sont en mode veille. Si la MDS active se termine de manière inattendue, un des MDS en attente devient active. En conséquence, les montages de clients continuent à fonctionner en cas de défaillance du serveur.

4. Disposition des fichiers et répertoires

CephFS permet aux utilisateurs de configurer les dispositions de fichiers et de répertoires pour utiliser plusieurs pools.

CephFS FUSE prend en charge des quotas sur n'importe quel répertoire d'un système. Le quota peut limiter le nombre d'octets ou le nombre de fichiers stockés sous ce point dans la hiérarchie des répertoires.

L'algorithme de Hachage uniforme organise l'intégralité de l'espace de hachage en anneau virtuel. Différent de l'algorithme de Hachage simple, on suppose que la valeur de l'espace d'un hachage H est entre 0 et $2^{32} - 1$.

The diagram illustrates a circular hash table with separate chaining. The table is represented as a circle with slots indexed from 0 to $2^{32}-1$. Slots 0, 1, 2, 3, and 4 contain linked lists of nodes. Node A points to slot 0, Node B to slot 1, Node C to slot 2, Node D to slot 3, and Node E to slot 4. Each node has a 'Hash()' function and a 'next' pointer to the next node in the chain.

En fonction de l'algorithme de Hachage, l'objet A sera attribué au noeud A, l'objet B sera attribué au noeud B, l'objet C sera attribué au noeud C et l'objet D sera attribué au noeud D.

Supposons que le noeud C est échoué, on peut voir que les objets A, B et D ne sont pas affectés et que seul l'objet C est déplacé vers le noeud D.

10

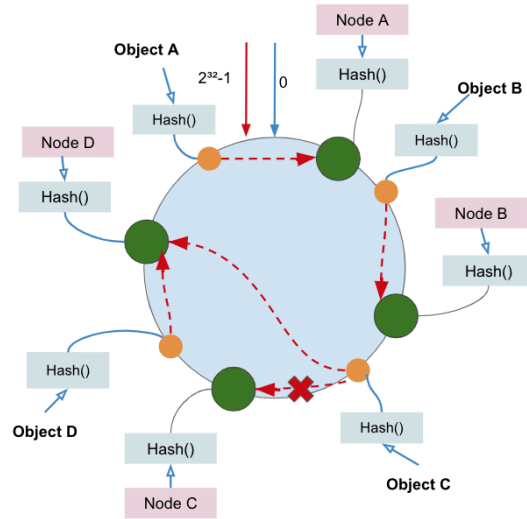


Figure 8: Suppression d'un noeud

Supposons qu'on ajoute un serveur (Node X) dans l'anneau de hachage. On peut constater que seul l'objet C a besoin de déplacer vers le nouveau noeud X.

En général, en cas d'ajout d'un nouveau serveur dans l'algorithme Hachage, les données affectées ne sont qu'entre le nouveau serveur et le serveur précédant (le premier serveur rencontré dans le sens inverse des aiguilles d'une montre).

2.2.2 Problème d'asymétrie des données dans l'anneau de hachage

Lorsque le nombre de serveur est petit, il est facile de générer un déséquilibre des données en raison d'une division inégale. C'est à dire, la majorité des objets stockés est placée sur un serveur. Comme montré dans la figure ci-dessous :

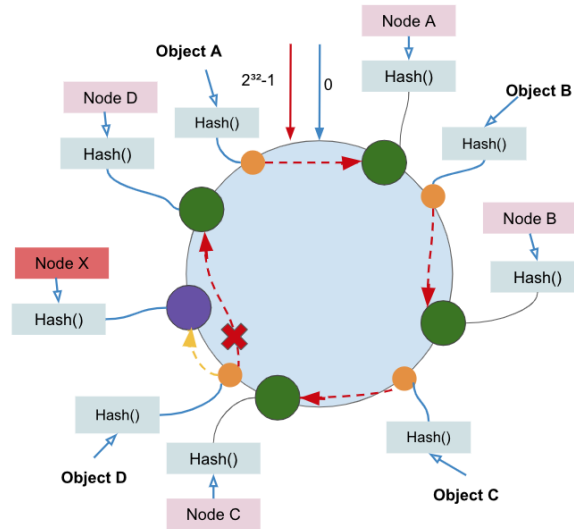


Figure 9: Ajout d'un noeud

Une grande quantité de données est concentrée sur le noeud A et une petite quantité sur le noeud B en même temps. Dans ce cas, si le noeud A est échoué, le calcul et le déplacement des données seront énorme. Pour résoudre ce problème, l'algorithme Hachage ajoute le mécanisme de noeud virtuel. C'est à dire qu'il calcule plusieurs hachages pour chaque noeud physique et met un noeud virtuel dans les positions correspondantes aux résultat de calcul, appelé aussi un noeud de service.

Par exemple, pour la situation précédente, on calcule 3 noeuds virtuels pour chaque noeud physique : NodeA#1, NodeA#2, NodeA#3, NodeB#1, NodeB#2, NodeB#3; pour former 6 noeuds virtuels.

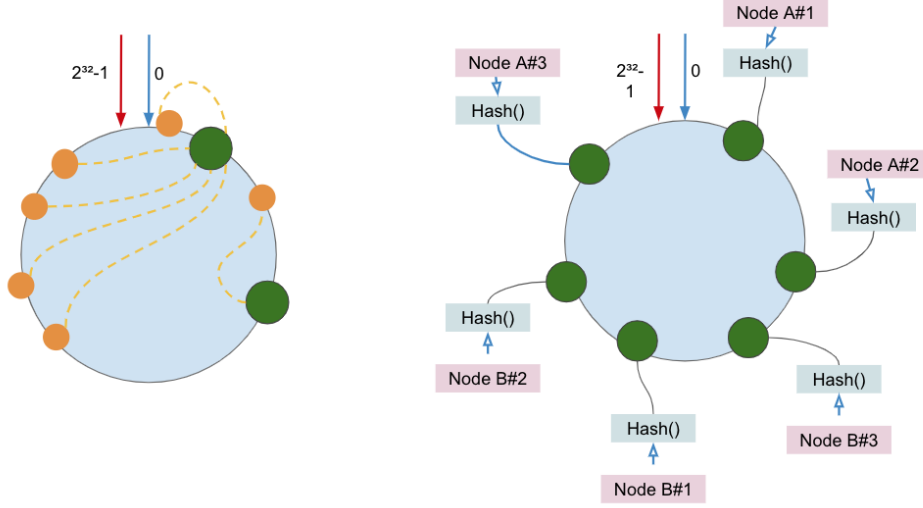


Figure 10: Transformation de noeuds physiques à noeuds virtuels

Dans le même temps, l'algorithme de positionnement des données n'a pas changé, mais il n'y a qu'une étape pour mapper le noeud virtuel sur le noeud physique. Dans les applications réelles, le nombre de noeud virtuel est généralement fixé à 32 ou plus, de sorte que même un petit nombre de noeuds physique peut réaliser une distribution de données relativement uniforme.

2.2.3 Hachage dans Ceph

Pour le placement des objets, Ceph utilise l'algorithme CRUSH, qui est basé sur l'algorithme Hachage. La PG de CRUSH est précisément le noeud virtuel dans l'algorithme de hachage uniforme en fixant le nombre de noeud virtuel (PG). Cependant, afin de trouver un OSD pour placer les objets, Ceph a besoins de 2 étapes principales :

1. Obtenir l'identificateur de PG

$$PGID = HASH[OBJID] \% PGNOMBRE$$

2. Mapper de PG (noeud virtuel) et de noeud physique

En utilisant la fonction `HASH crush_hash32(CRUSH_HASH_DEFAULT, value)`, on a simulé une situation de placement des objets dans 32 placements groupes. Pour valider l'homogénéisation, on a calculé les écart-types et les espérances à partir de 100 objets jusqu'à 10 000 000 objets, dont le pas est 100 entre 100 et 10 000, 10 000 entre 10 000 et 10 000 000. Les résultats d'une exécution est en ci-dessous :

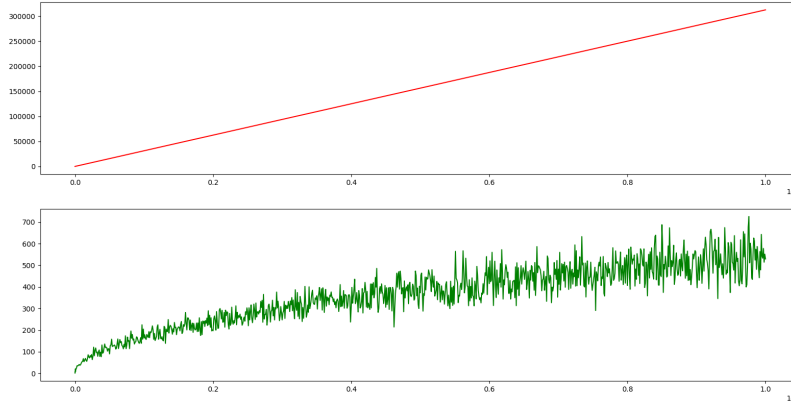


Figure 11: Les espérances (en haut) et les écart-types (en bas) de distribution d'objet

On peut voir que les espérances sont linéaires et que les écart-types sont dans une forme logarithmique. Ainsi l'écart-type se croiserait moins rapidement et la distribution serait presque homogène.

La première étape est le problème résolu par l'algorithme de hachage uniforme et la deuxième est à laquelle l'algorithme CRUSH doit répondre.

2.3 Algorithme CRUSH

Comme dit précédemment, l'objectif de l'algorithme CRUSH est d'obtenir une liste des OSDs :

$$CRUSH(PGID) = [OSD_1, OSD_2, OSD_3, \dots]$$

Ceph décrit toutes ses ressources matérielles du système comme une architecture arborescente, et puis génère une arborescence logique sous la forme de cluster Map conformément à certaines règles de tolérance aux pannes basées sur cette structure.

Supposons une structure comme celle indiquée dans la figure ci-dessous : Les bandes bleues en bas peuvent être vues comme des serveurs; les cylindres à l'intérieur comme des OSDs; les bandes violettes comme des cabinets (armoires) qui contiennent plusieurs serveurs; les bandes vertes comme une rangée de cabinet et la bande orange comme la racine de la structure dans le sens pratique.

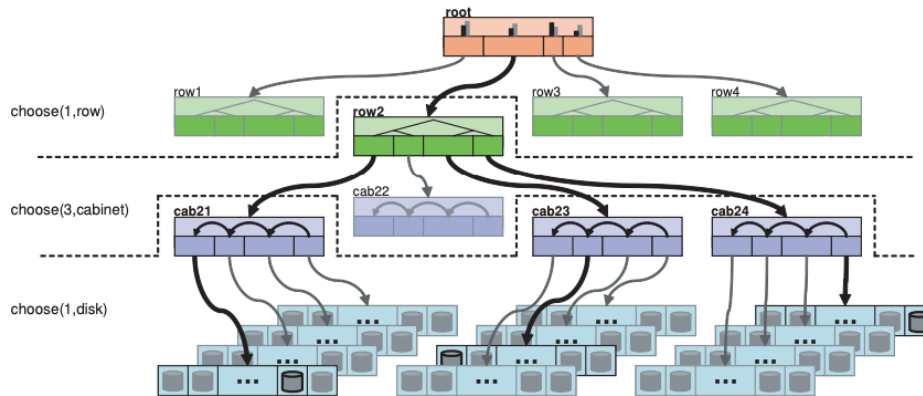


Figure 12: Une structure hiérarchique de 4 niveaux constitué par lignes, armoires et étalage de disques

Dans Ceph, toutes les bandes sont vues en tant qu'un noeud. Et généralement, Ceph considère deux type de noeuds.

1. Bucket : tous les noeuds logiques. Dans Ceph, un bucket est l'unité de domaine de l'isolation de défaillance. Par exemple, supposons que tous les cabinets ne partagent pas de circuit électrique, si l'alimentation d'un cabinet est en panne, il ne va pas affecter des autres cabinets.
2. OSDs : un OSD est un noeud de type réel. En général, un OSD n'est pas utilisé pour un domaine de l'isolation de défaillance.

2.3.1 Cluster Map

Cluster Map est composée de périphériques et de buckets, qui possèdent des descriptions des données et des poids. Un bucket peut contenir plusieurs périphériques ou d'autres buckets, cela lui permet de fournir une structure hiérarchique de stockage.

Les contenus de Cluster Map comprennent :

1. Epoch : le numéro de la version qui est une séquence croissante monotone. Plus l'Epoch est élevé, plus la version de CRUSH Map est récente. Ce mécanisme permet de comparer les différentes versions contenues par les OSDs ou les Clients pour déterminer la version qui doit être respectée. Autrement dit, lorsque la différence de valeur de l'Epoch des deux parties pendant la communication, il faut d'abord synchroniser la version de CRUSH Map la plus récente par défaut, puis les opérations suivantes sont effectuées.
2. Les adressages réseaux des OSDs.
3. L'état des OSDs. L'état des OSDs considère deux dimensions : si l'OSD travaille normalement (up/down) et si l'OSD est contenu dans au moins un PG (in/out). Du coup, chaque OSD a quatre états possibles.
 - up et in : cela indique que l'OSD fonctionne normalement et qu'il a déjà été contenu dans au moins un PG.
 - up et out : cela indique que l'OSD fonctionne normalement mais qu'il n'est pas contenu par un PG.
 - down et in : cela indique le fonctionnement anormal de l'OSD qui stocke des données.
 - down et out : cela indique que l'OSD a complètement échoué et ne comporte plus de PG.
4. Les paramètres de configuration de l'algorithme CRUSH. Il montre la hiérarchie du Cluster Ceph et les règles d'emplacements.

Algorithm 1 CRUSH placement for object x

```

1: procedure TAKE( $a$ )           ▷ Put item  $a$  in working vector  $\vec{i}$ 
2:    $\vec{i} \leftarrow [a]$ 
3: end procedure

4: procedure SELECT( $n, t$ )       ▷ Select  $n$  items of type  $t$ 
5:    $\vec{o} \leftarrow \emptyset$        ▷ Our output, initially empty
6:   for  $i \in \vec{i}$  do             ▷ Loop over input  $\vec{i}$ 
7:      $f \leftarrow 0$              ▷ No failures yet
8:     for  $r \leftarrow 1, n$  do     ▷ Loop over  $n$  replicas
9:        $f_r \leftarrow 0$          ▷ No failures on this replica
10:       $retry\_descent \leftarrow false$ 
11:      repeat
12:         $b \leftarrow bucket(i)$    ▷ Start descent at bucket  $i$ 
13:         $retry\_bucket \leftarrow false$ 
14:        repeat
15:          if “first  $n$ ” then     ▷ See Section 3.2.2
16:             $r' \leftarrow r + f$ 
17:          else
18:             $r' \leftarrow r + f_r n$ 
19:          end if
20:           $o \leftarrow b.c(r', x)$    ▷ See Section 3.4
21:          if  $type(o) \neq t$  then
22:             $b \leftarrow bucket(o)$    ▷ Continue descent
23:             $retry\_bucket \leftarrow true$ 
24:          else if  $o \in \vec{o}$  or  $failed(o)$  or  $overload(o, x)$ 
25:            then
26:               $f_r \leftarrow f_r + 1, f \leftarrow f + 1$ 
27:              if  $o \in \vec{o}$  and  $f_r < 3$  then
28:                 $retry\_bucket \leftarrow true$    ▷ Retry
29:                collisions locally (see Section 3.2.1)
29:              else
30:                 $retry\_descent \leftarrow true$    ▷ Otherwise
31:                retry descent from  $i$ 
32:              end if
33:            end if
34:            until  $\neg retry\_bucket$ 
35:            until  $\neg retry\_descent$ 
36:             $\vec{o} \leftarrow [\vec{o}, o]$            ▷ Add  $o$  to output  $\vec{o}$ 
37:          end for
38:        end for
39:       $\vec{i} \leftarrow \vec{o}$            ▷ Copy output back into  $\vec{i}$ 
40:    end procedure

41: procedure EMIT           ▷ Append working vector  $\vec{i}$  to result
42:    $\vec{R} \leftarrow [\vec{R}, \vec{i}]$ 
43: end procedure

```

Figure 13: Algorithme CRUSH

2.3.2 Le processus

Comme dit précédemment, la couche sous-jacente de Ceph est RADOS, qui est destinée au stockage des objets. Du coup, toutes les données vont d’abord être divisées en plusieurs objets en fonction d’une taille fixé (4M par défaut). Chaque objet possède un identifiant (oid) qui respecte les règles de nomination. Avec l’identifiant de l’objet, Ceph peut ensuite calculer l’identifiant de PG pour obtenir le pgid. Et puis lister une table des OSDs pour exécuter l’algorithme CRUSH. Suivant il faut filtrer les OSDs afin de trouver les OSDs satisfaits des demandes requises. Par exemple, les OSDs doivent marcher normalement. Finalement, le client déclenche une demande au premier OSD dans la table qui va faire le traitement correspondant. En résumé, la fonctionnement de l’algorithme CRUSH est d’obtenir une table des OSDs (avec un OSD primaire et d’autres OSDs de répliquât).

L’algorithme CRUSH calcule la distribution des objets de données en fonction du poids de chaque machine. La distribution des objets est déterminée par le Cluster Map et la stratégie de distribution des données. Cluster Map décrit les ressources de stockage disponibles et la hiérarchie telle que le nombre de racks, le nombre de serveurs dans chaque racks et le nombre de disques sur chaque serveur. La stratégie de distribution des données consiste en des règles d’emplacement, qui détermine le nombre de copie de chaque

objet de données et les limites de ces copies stockées. Par exemple, 3 copies sont placées dans des racks différents. Cela permet de réaliser l'isolation des domaines de défaillance parce que les 3 copies ne partagent pas un même circuit physique.

2.3.3 Fonction SELECT

Un exemple de règle de Ceph conformément à la figure 11 :

```
take(root)
select(1,row)
select(3,cabinet)
select(1,osd)
```

Le but de la fonction SELECT est d'élire n buckets parmi les noeuds fils du bucket courant dans l'arborescence. En effet, rappelons que le but de l'algorithme est de parcourir l'arborescence de façon à choisir les OSDs désirés.

Chaque bucket a la responsabilité de déterminer la branche qui sera parcourue. Il existe plusieurs méthodes d'élection:

1. **Uniform Bucket** : Items de poids égaux
2. **List Bucket** : Choix du plus récent possible dont le poids dépasse la somme des poids des autres items dans le bucket
3. **Tree Bucket** : Similaire à la List Bucket mais avec un arbre binaire de recherche.
4. **Straw Bucket** : Un nombre aléatoire est tiré dans l'intervalle $[0; poids]$ et les items ayant le résultat le plus élevés sont élus. Ainsi, les items ayant le poids le plus élevé ont plus de chance d'être élu.
5. **Straw Bucket 2** : Optimisation du type précédent. Il s'agit d'un type apparu dans les dernières versions de Ceph.

Ainsi, après l'appel de la fonction TAKE à un noeud, la fonction SELECT est appelée : le noeud choisit n noeuds parmi ses enfants. La fonction SELECT est de nouveau appelée pour ces enfants choisis, jusqu'à l'obtention des OSDs qui seront les OSDs élus. Lors de l'appel de la fonction SELECT pour un noeud donné, le noeud utilise l'algorithme qui correspond à son type.

Par conséquent, lorsque nous voulons stocker un PG, nous pouvons choisir le nombre de répliques dont nous avons besoin. Étant donné que chaque noeud fils correspond à un lieu distinct, nous avons la certitude que les répliques seront répartis de façon à limiter les risques de perte de toutes les sauvegardes d'un coup (exemple : incendie qui touche une salle, on a la certitude qu'un réplica se trouve dans une autre salle).

2.4 Algorithme PAXOS

Supposons un ensemble de processus pouvant proposer des valeurs. Un algorithme de consensus s'assure qu'une seule parmi les valeurs proposées est choisie.

L'algorithme de Paxos est un algorithme de consensus basé sur le passage de message et très tolérant aux pannes, proposé par Leslie Lamport en 1990. Il est une famille de protocoles permettant de résoudre le consensus dans un réseau de nœuds faillibles qui sont susceptibles d'avoir des pannes. Il permet aux différentes machines de proposer des valeurs. Le but est que chacune de ces machines choisisse une et une seule valeur. La valeur choisie était proposée par l'une des machines. Une fois qu'un consensus est atteint, il ne peut pas évoluer vers un autre consensus.

2.4.1 Algorithme PAXOS original

1. Rôles

Paxos distingue 5 rôles distincts :

(a) Client

Le Client fait des requêtes aux systèmes distribués, et attend une réponse. Par exemple une requête d'écriture sur un système de fichiers distribué.

(b) Acceptor (Votants)

Les Acceptors servent de mémoire résistante aux pannes. Les Acceptors sont regroupés en groupes nommés Quorums. Chaque message envoyé à un Acceptor doit l'être au Quorum entier. Un message reçu sur un Acceptor qui n'a pas été reçu sur tous les autres Acceptors du Quorum est ignoré.

Quorum : un quorum est constitué de la majorité des accepteurs participants.

(c) Proposer

Un Proposer pousse la requête du client, il a pour but de convaincre les Acceptors de tomber d'accord, et agit comme coordinateur pour avancer quand un conflit se présente.

(d) Learner

Les Learners servent à la réplication. Une fois qu'une requête d'un Client a été acceptée par les Acceptors, le Learner peut agir (i.e.: exécuter une requête et envoyer la réponse au client). Pour augmenter la disponibilité on peut ajouter des Learners.

(e) Leader

Paxos nécessite un Proposer différent (appelé le leader) pour avancer. Plusieurs processus peuvent croire être le leader, mais le protocole ne garantit l'avancement que si l'un d'eux est choisi. Si deux processus croient qu'ils sont leader, ils peuvent bloquer le protocole en envoyant continuellement des propositions conflictuelles. Mais l'intégrité des données est toujours préservée dans ce cas.

2. Basic-PAXOS

- Phase 1 (prepare) : lorsque vous essayez de proposer une valeur, commencez par diffuser un appel rpc appelé prepare. Préparez rpc pour deux objectifs : 1. Recherchez la valeur qui a été sélectionnée (le cas échéant, nous l'utilisons dans la deuxième phase) 2. Bloquez l'ancienne proposition qui n'a pas été complétée pour empêcher l'ancienne proposition de rivaliser avec notre nouvelle proposition.
- Phase 2 (accept) : diffusez un autre appel rpc appelé accept pour laisser le système accuser réception d'une valeur spécifique. Une fois que plus de la moitié des accepteurs ont répondu "Acceptor" dans cet appel, nous pouvons être sûrs que l'offre est "choisie".
- Phase 3 (Learning) : les learners sont informés du résultat du tour.

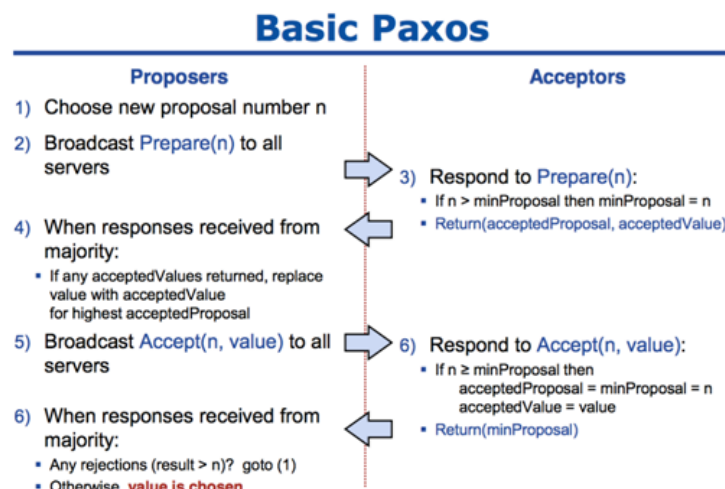


Figure 14: Basic Paxos Process

Notes :

Chaque proposition a un numéro unique et les numéros les plus élevés ont priorité sur les numéros les plus bas.

Seuls les proposants qui ont lancé la proposition savent quelle valeur a été sélectionnée. Par exemple, l'accepteur ne sait pas quelle valeur est sélectionnée. Si d'autres serveurs veulent savoir exactement quelle valeur est sélectionnée, le seul moyen est d'initier vous-même un processus de proposition paxos.

3. Multi-PAXOS

Les Paxos de base peuvent être utilisés pour déterminer un journal. Multi-Paxos exécute un protocole 2PC (Two-Phase Commit) Paxos pour chaque journal afin de déterminer plusieurs journaux, qui est un flux de journal. Avec le flux de consignment, un modèle "machine à état de réplication" peut être construit en fonction du flux de consignment.

4. Basic-PAXOS VS Multi-PAXOS

Le protocole de Basic-PAXOS est plus complexe et relativement inefficace. Alors maintenant, tous les protocoles liés au PAXOS doivent être basés sur le multi-paxos.

2.4.2 PAXOS dans CEPH

Ceph utilise Paxos dans le cadre des processus de moniteur (MON) pour déterminer quels OSD sont actifs et dans le cluster. Ceph a choisi d'implémenter sa propre version de Multi-Paxos pour s'assurer que le cluster Monitor fournit des services cohérents. Ceph Multi-Paxos utilise la modification de métadonnées supérieure sous la forme d'une proposition unique visant à s'étendre sur le cluster. Ceph utilise simplement Paxos pour faire référence à Multi-Paxos. Nous utilisons également cette référence.

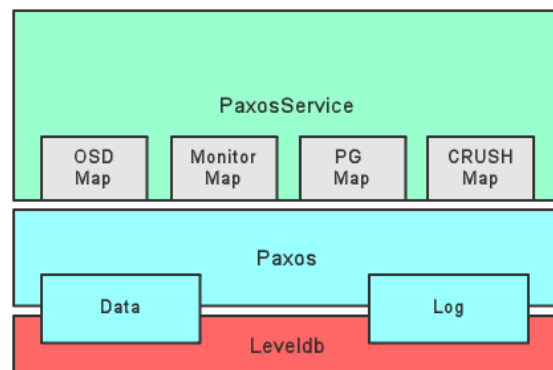


Figure 15: Ceph monitor Architecture

2.4.3 Algorithme de CEPH PAXOS

- Introduction

Le nœud Paxos est lié au nœud Moniteur et chaque moniteur démarre un Paxos. Lorsque la plupart des nœuds Paxos ont survécu, le cluster peut être exécuté : lors de la fourniture d'un service normal, un nœud joue le rôle de leader et les autres le rôle de Peon. Seul le leader peut initier une proposition et tous les rôles Peon acceptent ou rejettent la proposition du leader en fonction de l'histoire locale et répondent au leader. Leader compte et soumet les propositions acceptées par plus de la moitié des nœuds de Paxos.

Chaque proposition est un ensemble de modifications des méta-informations du moniteur, sérialisées et transmises dans la couche Paxos. Lorsque le leader de file initie la proposition et que Peon accepte la proposition, il écrit dans le log local, lequel est éventuellement écrit dans la base de données, et la proposition écrite dans la base de données est enfin visible. Dans l'implémentation, la même instance de

base de données est utilisée pour stocker le stockage du journal et des données finales, et le namespace est utilisé pour la distinction.

En plus des guides Leader et Peon mentionnés ci-dessus, le nœud Paxos peut également se trouver dans l'un des trois états Probing, Synchronizing et Election, comme illustré à la figure suivante. Parmi eux, Election est utilisé pour élire un nouveau leader. Probing est utilisé pour découvrir et mettre à jour les informations sur les nœuds du cluster, pour découvrir les différences de données entre les nœuds Paxos et pour suivre les données dans l'état Synchronizing. Lorsque l'appartenance change ou que le délai d'expiration du message ou du bail est dépassé, le nœud entre dans l'état de Probing par le biais de bootstrap et diffuse le message prob à d'autres nœuds. Tous les nœuds non prob ou synchronisés qui reçoivent le message reviennent à l'état de Probing. L'état de probing a reçu plus de la moitié des acceptions des membres et est entré dans l'état d'élection. Dans l'état de Probing, les nœuds avec des lacunes excessives dans les données entreront dans l'état de Synchronizing pour la synchronisation ou la synchronisation partielle.

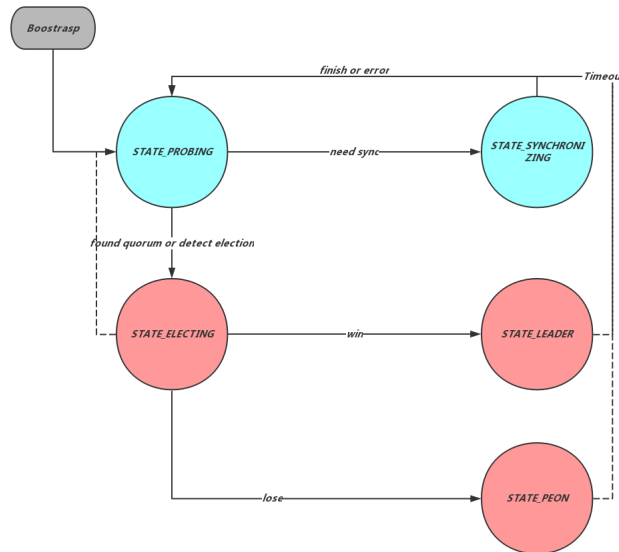


Figure 16: Ceph Paxos

Le chef envoie un message Lease à tous les Peons, et le Peon qui reçoit le Lease peut fournir le service de lecture Paxos directement avec des données locales pendant la durée du bail pour partager la pression de demande en lecture seule du leader.

Le leader sélectionnera le plus grand et unique numéro proposé dans le cluster actuel, appelé P_n . À chaque fois, le nouveau leader augmentera d'abord son propre P_n et l'utilisera pour se marquer en tant que scène du Leader, en tant qu'horloge logique dans l'algorithme Ceph Paxos. Dans le même temps, chaque proposition se voit attribuer une version globalement unique et croissante, utilisée comme emplacement d'index pour le log. P_n et Version seront transmis avec la communication du message entre Paxos afin que l'autre partie puisse juger le leader qui est nouveau ou ancien. Le nœud Paxos conservera le numéro de version de la plus grande proposition actuellement soumise par lui-même avec le journal pour une récupération ultérieure.

- Cas normal

Dans l'état de service normal, il existe un leader unique et un quorum confirmé : le leader encapsule chaque demande d'écriture dans une nouvelle proposition et l'envoie à chaque nœud de Quorum. Le processus est le suivant (le Quorum est fixé ici) :

- Le leader ajoute la proposition au journal local et envoie un message "début" à tous les nœuds du quorum, le message portant la valeur de proposition, P_n , et le `last_commit` pointant vers la version précédente de la proposition;

- Peon reçoit le message "begin" et ignore si "accept" est un pn supérieur, sinon la proposition est écrite dans le journal local et renvoie un message "accept". En même temps, Peon expirera le "Lease" actuel et ne fournira plus de services jusqu'à la prochaine fois que le "Lease" sera reçu;
- Peon reçoit le message "begin" et ignore si "accept" est un pn supérieur, sinon la proposition est écrite dans le journal local et renvoie un message "accept". En même temps, Peon expirera le "Lease" actuel et ne fournira plus de services jusqu'à la prochaine fois que le "Lease" sera reçu;
- Le leader reçoit un commit après avoir reçu tous les accepts de Quorum. Exécute l'élément Log dans la base de données locale, revient à l'appelant et envoie un message de Commit à tous les nœuds de quorum.
- Peon reçoit le message de Commit et l'exécute dans la base de données locale en complétant le Commit.
- Le leader ajoute un message de Lease pour amener tout le cluster à l'état actif.

- Election du leader

Le délai d'expiration de Peon's Lease ou tout délai de message du leader ramènera l'ensemble du cluster à l'état de Probing, qui déterminera les nouveaux membres et entrera finalement dans l'état Election. Chaque nœud sera maintenu localement et sélectionnera de manière interactive le numéro de tour principal election_epoch, qui augmentera de façon monotone et sera incrémenté au début et à la fin de la sélection et pourra donc être jugé en fonction de sa parité. Le processus de sélection est le suivant :

- Ajoutez 1 à election_epoch pour envoyer un message de proposition à tous les autres nœuds de Monmap;
 - Le nœud qui a reçu le message Propose entre dans l'état d'élection et répond à l'Ack uniquement pour l'élection mise à jour et la valeur de classement est supérieure à son propre message. Le rang ici est simplement déterminé par la taille de l'ip. Chaque nœud ne fait Ack qu'une seule fois dans chaque election_epoch, ce qui garantit que le leader final doit avoir le support de la plupart des nœuds;
 - Le nœud qui envoie Propose compte le nombre d'acquittements reçus. Après avoir reçu la plupart des acks de Monmap, il peut entrer dans le processus de victoire (les nœuds qui envoient des ACK sous forme de quorum), et election_epoch incrémente. Il termine la phase d'élection et envoie la victoire à tous les nœuds du quorum pour les informer de l'époque et du quorum actuel, enfin il entre dans l'état Leader;
 - Le nœud qui reçoit le message VICTORY termine l'élection et entre dans l'état Peon.
- #### - Récupération

Après la phase d'Election susmentionnée, les membres Leader, Peon et Quorum ont été identifiés. Cependant, en raison de la stratégie de la phase Election, le nouveau leader ne dispose pas nécessairement de données complètes et validées. Par conséquent, il est nécessaire de passer par la phase de récupération avant le début effectif de la lecture et de l'écriture cohérentes. Il convient de noter que la proposition de restriction Ceph Paxos est lancée dans l'ordre de version. La proposition précédente est "commit" avant que la suivante ne puisse être lancée, c'est-à-dire qu'il peut y avoir au plus une donnée non validée dans Recovery. Bien que cette approche sacrifie les performances, elle simplifie grandement la mise en œuvre de la phase de récupération et de la réalisation de l'algorithme de cohérence. Le sacrifice de ces performances peut être compensé par l'agrégation de la couche Ceph.

- Leader génère un nouveau Pn plus grand et l'envoie à tous les Peons via un message de Collect.
- Le péon reçoit le message de collecte et ne l'accepte que lorsque Pn est supérieur au maximum Pn qu'il a accepté. Peon renvoie les informations de journal du commit plus que Leader par le dernier message et les données non validées;
- Le chef reçoit le message "Last", met à jour ses données de Commit et envoie les nouvelles informations du journal de Commit à tous les péons devant être mis à jour via le message de Commit;
- Après réception du "Last" message de "Peon accept", si la grappe contient des données non validées, la proposition est soumise à nouveau en premier, ou un message de Lease est envoyé au peon pour actualiser son bail;

On peut voir que lorsque la distance entre le leader et le péon est grande, le temps nécessaire pour extraire et rejouer le journal est très long, c'est pourquoi Avant de commencer l'élection, Ceph Monitor participe pour la première fois à la synchronisation, comme le montre la figure précédente. L'intervalle d'informations de journalisation du nœud Paxos est réduit à un intervalle suffisamment petit. Cette longueur est configurée par `paxos_max_join_drift`, dont la valeur par défaut est 10.

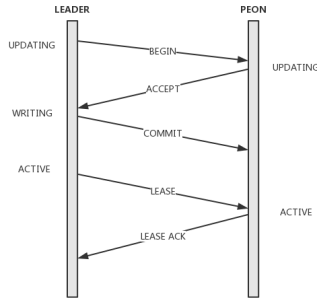


Figure 17: Paxos - Cas normal

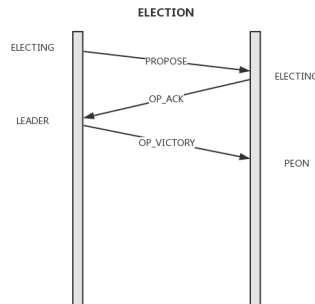


Figure 18: Paxos - Election

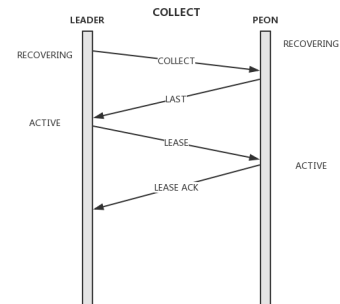


Figure 19: Paxos - Récupération

- Changement d'adhésion

Les informations sur les membres de Ceph Paxos sont enregistrées dans Monmap, qui peut être initialisée à partir du fichier de configuration ou ajoutée ou supprimée ultérieurement. Ceph Monitor introduit la phase de "Probing" pour la mise en œuvre des modifications de Membership : démarrage de nœud, nouvelle jonction de nœud, délai Paxos écoulé, nouveaux messages prob et informations Monmap modifiées, et Bootstrap utilisé pour entrer l'état du Probing. Dans cet état, le nœud de surveillance s'envoie un message prob afin de détecter l'existence de l'autre partie afin de générer et d'interagir avec les informations Monmap. Au cours du processus, l'ensemble du cluster Paxos a cessé de fournir des services.

- Compaction du log

Ceph introduit un mécanisme pour supprimer les anciennes données de journal. Après chaque validation réussie, Monitor vérifiera la quantité actuelle de données du journal, qui sera tronquée après une certaine valeur de configuration. La longueur de cette réservation est contrôlée par `paxos_min`. La valeur par défaut est 500.

2.4.4 Sélection consensus de CEPH PAXOS

- State Machine System

Ceph Monitor a choisi State Machine System au lieu de Primary Backup System. Le contenu stocké dans le Log et les données d'interaction entre les nœuds Paxos sont des opérations idempotentes telles que Put, Erase et Compacat. Elles sont en réalité écrites sur la state machine après la validation.

- Une seule proposition à la fois

La proposition de Ceph Paxos a été initiée de manière stricte et ordonnée et seule la précédente a été "Log Commit" pour initier une nouvelle proposition. Cela garantit que le cluster ne peut disposer qu'une proposition "uncommitted". Cela simplifie aussi la logique de mise en œuvre de Recovery. C'est faisable parce que l'agrégation des couches supérieures de Ceph Monitor et d'autres mécanismes pour réduire la mise en œuvre du protocole consensus réduit considérablement les exigences de performances de Ceph Paxos.

- Quorum fixe (Designated)

Pour l'algorithme de Paxos, quel que soit le processus de sélection ou le processus d'accès normal, il est nécessaire de garantir le succès de la plupart des nœuds (Quorum). En général, ce quorum n'est pas fixé à chaque fois, et Ceph Paxos choisit de déterminer une collection de quorum dès que la sélection de leader est réussie. Toutes les opérations suivantes sont uniquement envoyées à ce nœud et attendent

la réponse de tous les noeuds de la collection. Tout délai dépassant sera renvoyé à l'état "Probing" au cours du processus "bootstrap".

- Recovery bidirectionnelle

Étant donné que la stratégie de sélection de Ceph Paxos ne sélectionne le leader qu'en fonction de la taille de l'adresse du noeud, le noeud qui devient le leader peut ne pas disposer des données les plus récentes. Par conséquent, le Leader doit restaurer ses données et celles du cluster à l'étape "Recovery" avant de fournir le service. La direction des données de "Recovery" inclut de Peon à Leader et de Leader à Peon.

- Utiliser "Lease" pour optimiser les demandes en lecture seule

Ceph Paxos a introduit le mécanisme "Lease" pour supporter que Peon peut partager la pression de Leader. Pendant la période "Lease", Peon peut utiliser les données locales pour traiter les demandes en lecture seule; Peon annulera le "Lease" local lors de la réception d'une nouvelle proposition. Après la proposition "commit" ou le délai du "Lease" du Leader, le Leader actualisera le "Lease" de tous les Peon.

- Leader Peon détecte le lancement simultané de nouvelles élections

Le message "Lease" entre Leader et Peon assume également la tâche de détection de survie. Cette détection est bidirectionnelle. Une nouvelle série de "Election" va déclencher si Leader ne reçoit pas "Lease ACK" de Peon ou Peon n'a pas reçu d'actualisation de la part de Leader après la fin du délai de "Lease".

2.5 L'intérêt du système, concurrence, avantage du Ceph par rapport aux autres systèmes

Les avantages de Ceph sont variés :

1. Algorithme CRUSH (Controlled Replication Under Scalable Hashing)

Ceph a abandonné le schéma traditionnel d'adressage des méta-données de stockage centralisé et a appliqué comme alternative l'algorithme CRUSH pour mener à bien l'opération d'adressage des données. En se basant sur la cohérence de hashing, CRUSH prend en compte l'isolement des domaines de reprise et implémente des règles de positionnement des réplicas, telle que la reconnaissance de la salle des machines et du rack.

2. La haute disponibilité

L'administrateur de Ceph peut définir le nombre de copies de données. En même temps, l'algorithme CRUSH peut être utilisé pour spécifier l'emplacement de stockage physique de la copie afin de séparer le domaine de pannes, en prenant en charge la cohérence de données. Ceph peut tolérer plusieurs situations de panne et tenter d'auto-réparer en parallèle. Les noeuds de stockage de Ceph peuvent être gérés et réparés automatiquement. En plus, il n'y a pas de point de défaillance unique.

3. La haute performance

Comme il s'agit de copies multiples, Ceph peut être hautement parallélisé lors des opérations de lecture et d'écriture. Dans un autre point, le client de Ceph interagit avec les équipements de stockage (OSD) sur la lecture ou l'écriture des données. Aucun serveur de métadonnées n'est requis dans le stockage de bloc et le stockage d'objet.

4. La haute extensibilité

Comme Ceph n'a pas de point maître, ça sera plus facile de l'étendre. Théoriquement, il peut prendre en charge des milliers de noeuds de stockage.

5. La richesse des caractéristiques

Ceph supporte 3 types d'interfaces d'appel : le stockage d'objet, le stockage de bloc et le système de fichier compatible POSIX. Ceph propose des scénarios différents de stockage en répondant à la variété de demandes de stockage.

3 Évaluation des performances (cas général de Ceph)

3.1 Les métriques pertinentes et les paramètres à faire varier

Pour mettre en place une étude de performance sur notre architecture Ceph, nous avons choisis certaines métriques pertinentes à évaluer :

1. IOPS

Input/Output Operations Per Second est une performance de mesure commune utilisée pour référencer les appareils de stockage informatiques et donc bien-sûr pour Ceph aussi.

2. BandWidth

BandWidth est synonyme de taux de transfert de données, c'est-à-dire le volume de données pouvant être transporté d'un point à un autre dans un laps de temps donné (généralement une seconde).

3. Latency

On parle de latency pour le temps d'accès à une information sur notre système de stockage.

Et aussi les paramètres à faire varier :

1. Block size

La taille d'un bloc dans les systèmes de stockage de données et de fichiers.

2. Queue Depth

Nous pouvons optimiser la longueur de la file d'attente des disques qui sont ajoutés aux instances après le déploiement afin d'accroître les performances d'entrée-sortie de ces disques.

3.2 Étude de notre cas (Snapshot des résultats dans l'annexe A)

- Évaluation de la performance de base : réseau et disque

Fondamentalement, l'analyse de la performance est une question de comparaison. Par conséquent, avant de commencer l'analyse de notre cluster, nous devons obtenir des statistiques de performance de base pour les deux composants principaux de notre infrastructure Ceph : les disques et le réseau.

- Disque

On fait le test avec l'outil dd : Notez la dernière statistique fournie, qui indique les performances du disque en Mo/s.

Résultat : Lecture : 751 Mb/s; Ecriture : 231 Mb/s

- Réseau

On fait le test avec l'outil iperf pour tester le débit du réseau. Notez les statistiques de bande passante en Mbits/s, car elles indiquent le débit maximal pris en charge par votre réseau.

Notre bande passante : 15 Mb/sec

- Benchmark d'un cluster de stockage Ceph avec bench

Ceph inclut la commande de rados bench, conçue spécifiquement pour évaluer un cluster de stockage RADOS.

Paramètres : 10 secondes de test, threads=16, block-size=4MB

Résultat (On choisit quelques métriques pertinentes qu'on s'intéresse):

- Écriture :

Écriture totale : 292

Bande passante : 114.889 Mb/sec (231 Mb/s)

IOPS moyen : 28

Latence moyenne : 0.556s

- Lecture séquentielle
Lecture totale : 292
Bande passante : 450.994 Mb/sec (751 Mb/s)
IOPS moyen : 112
Latence moyenne : 0.139s
- Lecture aléatoire
Lecture totale : 1013
Bande passante : 390.121 Mb/sec (751 Mb/s)
IOPS moyen : 97
Latence moyenne : 0.163s

Analyse : Par rapport aux performances de notre disque, Le cluster Ceph peut atteindre la moitié de la limite de lecture et d'écriture du disque. Le nombre d'IOPS est petit (Probablement parce que la taille de bloc est de 4M) mais la bande passante est très élevée. Les lectures aléatoires effectuent plus de lectures que les lectures séquentielles.

- Benchmark d'un Ceph Block Device avec fio

On teste le Block Device (BD) du Ceph en utilisant fio, un outil de test de contrainte d'E/S utilisé pour tester les performances d'E/S de disque.

Paramètres : 1Go des choses à E/S, IODepth=32, block-size=4k

Résultat (On choisit quelque métriques pertinentes qu'on s'intéresse):

Pour faire la comparaison des résultats, on a aussi testé la performance de notre périphérique du disque de ce noeud (/dev/sda) et les résultats entre parenthèses.

- Écriture aléatoire:
IOPS : 225 (6820)
Bande passante : 923 kB/s (26.6 MB/s)
Latence moyenne : 141.95 ms
- Lecture aléatoire:
IOPS : 1880 (897)
Bande passante : 7524 kB/s (3589kB/s)
Latence moyenne : 17.001 ms

Analyse : Dans notre cas, les performances de lecture du périphérique ceph block sont meilleures que celles du disque dur, mais les performances en écriture sont inversées. Au cours du processus d'écriture aléatoire du test, il provoque la désactivation de l'OSD. Du coup je pense que l'écriture de BD de notre Ceph est anormal, il existe quelque problèmes. Ou peut-être que la raison suivante a causé cette exception :

- La différence entre le disque dur réel et le périphérique de test (100Go et 1Go)
- Quelques paramètres différents
- Considérez que notre taille de bloc est de 4k et que nous écrivons des fichiers 1Go. Cela signifie que nous avons écrit beaucoup de petits fichiers. Donc, ceph peut avoir de mauvaises performances lors de l'écriture de nombreux petits fichiers.

- Benchmark d'une Ceph Object Gateway avec bench-swift

L'outil swift-bench teste les performances du cluster Ceph en simulant les demandes PUT et GET des clients et en mesurant leurs performances.

Swift-bench mesure la performance en nombre d'objets/s. On peut convertir cela en Mo/s, en multipliant par la taille de chaque objet :

Résultat : PUTS : environs 700 Ko/s; GETS : environs 1.3 Mo/s

Analyse : Par rapport aux performances de notre réseau (1.875 Mo/sec), le GETS peut atteindre la limite de bande passante de près de 2/3.

4 Réalisation

Pour construire un cluster de Ceph, il faut au moins un serveur moniteur (MON) et deux serveurs de stockage d'objet (OSD). Pour tester la robustesse, on a choisi de construire un cluster de 3 serveurs, il y a 3 OSD et 3 MON, 1 serveur de méta-données (MDS) pour CephFS, et un serveur de manager (MGR). En plus, on voudrait ouvrir un serveur de stockage d'objet de RADOS (RGW).

On a construit un cluster de 3 machines en utilisant 3 VPS de Digital Ocean, dont les systèmes sont Ubuntu 18.04 avec le noyau 4.15.0 et Ceph version mimic. Ils sont nommés:

Droplet Name: SR05-1
IP Address: 46.101.122.175
Droplet Name: SR05-2
IP Address: 157.230.115.67
Droplet Name: SR05-3
IP Address: 134.209.226.112

Les trois serveurs sont exposés dans les réseaux publics avec les adresses IP ci-dessus. Ils sont également connectés dans un réseau privé 10.135.0.0/16. Grâce à ces connexions, il y a moins de latence et plus de sécurité. Ce qui est aussi important, c'est de limiter l'usage de la bande passante. En effet, les débits dans les réseaux privés de Digital Ocean ne consomment pas de bande passante.

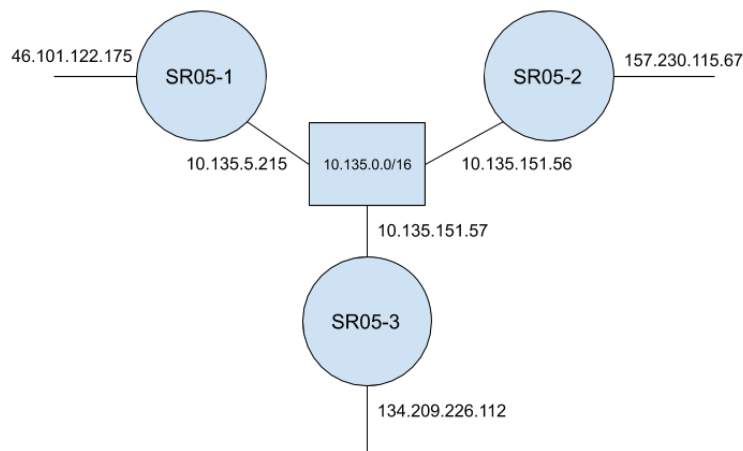


Figure 20: Topologie

Afin de faciliter la connexion, on a configuré le nom d'hôte dans chaque serveur. Voici la configuration dans le serveur SR05-1 `/etc/hosts`:

```
# Your system has configured 'manage_etc_hosts' as True.
# As a result, if you wish for changes to this file to persist
# then you will need to either
# a.) make changes to the master file in /etc/cloud/templates/hosts.debian.tmpl
# b.) change or remove the value of 'manage_etc_hosts' in
#     /etc/cloud/cloud.cfg or cloud-config from user-data
#
#127.0.1.1 SR05-1 SR05-1
127.0.0.1 localhost

# The following lines are desirable for IPv6 capable hosts
::1 ip6-localhost ip6-loopback
fe00::0 ip6-localnet
```

```
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
ff02::3 ip6-allhosts
```

```
10.135.5.215 SR05-1
10.135.151.56 SR05-2
10.135.151.57 SR05-3
```

Chaque serveur a un disque supplémentaire nommé sda. Les disques durs sont utilisés par OSD pour stocker les objets. Le disque dur de SR05-1 est de 100Go, et 50Go pour les autres. Ainsi on a au total 200Go d'espace de stockage pour les 3 OSDs et pour notre cluster. Ainsi les poids sont 0,098 pour osd.0, 0,049 pour osd.1 et osd.2.

La vue globale de notre cluster est générée par la commande "ceph -s".

```
cluster :
  id:      407124ce-2e7c-45c0-878a-b93116505e1a
  health: HEALTHOK

services :
  mon: 3 daemons, quorum SR05-1,SR05-2,SR05-3
  mgr: SR05-3(active)
  mds: inoki-cephfs-1/1/1 up {0=SR05-1=up:active}, 1 up:standby
  osd: 3 osds: 3 up, 3 in
  rgw: 1 daemon active

data:
  pools:      8 pools, 52 pgs
  objects:    243 objects, 44 KiB
  usage:      5.1 GiB used, 195 GiB / 200 GiB avail
  pgs:        52 active+clean
```

Il y a trois moniteurs donc 3 quorums. Le gestionnaire (manager) est dans SR05-3. En plus, le dashboard est aussi dans SR05-3, et accessible dans cet URL <https://134.209.226.112:9600/>.

Pour utiliser CephFS, on a ouvert 2 Metadata Serveurs en SR05-1 et SR05-3.

Afin d'essayer Ceph RADOS Object Stockage, un Gateway dans SR05-1, qui est compatible avec Amazon S3 est ici: <http://46.101.122.175:7480/>.

Le cluster mapping est ci-dessous:

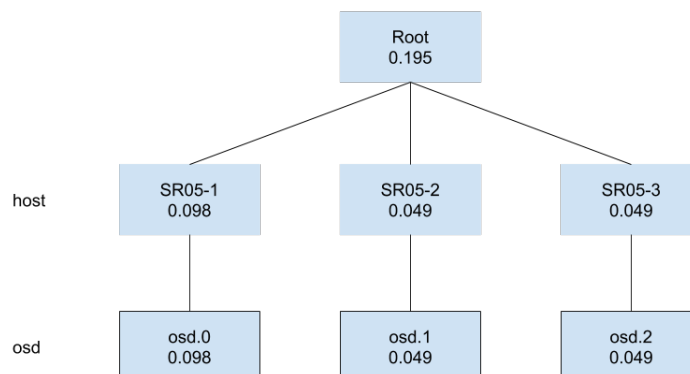
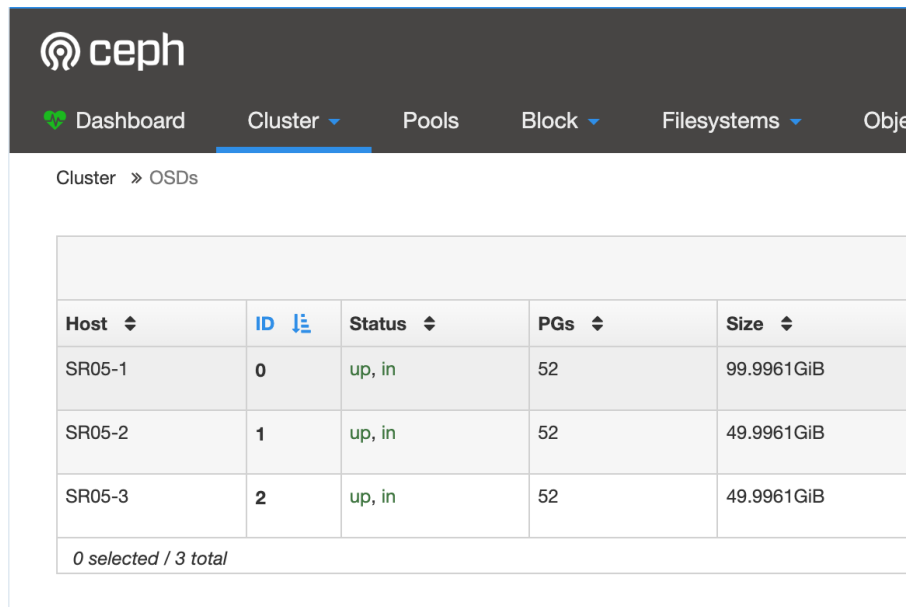


Figure 21: Couche de CRUSH map

Pour établir le mapping entre Placement Group, l'algorithme straw2 est utilisé dans tous les noeuds en couches host et root. La règle est de choisir un osd dans tous les hosts. Donc le résultat doit être, pour

chaque Placement Group, on a un réplica dans chaque osd. Donc pour chaque OSD, il y aura 52 PGs. Pour vérifier, on a trouvé cette page dans le dashboard :



| Host | ID | Status | PGs | Size |
|--------|----|--------|-----|------------|
| SR05-1 | 0 | up, in | 52 | 99.9961GiB |
| SR05-2 | 1 | up, in | 52 | 49.9961GiB |
| SR05-3 | 2 | up, in | 52 | 49.9961GiB |

0 selected / 3 total

Figure 22: PG OSD map

Pour vérifier RGW, on a créé 3 petits programmes en Python, en utilisant boto S3 API.

```
# create-bucket.py
import boto
import boto.s3.connection
access_key = '*'
secret_key = '*'

conn = boto.connect_s3(
    aws_access_key_id = access_key,
    aws_secret_access_key = secret_key,
    host = 'ceph.inoki.cc',
    port = 7480,
    is_secure=False,
    calling_format = boto.s3.connection.OrdinaryCallingFormat(),
)

bucket = conn.create_bucket('inoki-bucket-3')
```

Ce programme sert à créer un bucket dans RADOS Object Storage.

```
# list-bucket.py
import boto
import boto.s3.connection
access_key = '*'
secret_key = '*'

conn = boto.connect_s3(
    aws_access_key_id = access_key,
    aws_secret_access_key = secret_key,
    host = 'ceph.inoki.cc',
    port = 7480,
    is_secure=False,
```

```

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

buckets = conn.get_all_buckets()

for bucket in buckets:
    print("{name}\t{created}".format(
        name = bucket.name,
        created = bucket.creation_date,
    ))
if len(buckets) == 0:
    print("No bucket")

```

Ce programme sert à afficher tous les buckets dans RADOS Object Storage.

```

# create-object.py
import boto
import boto.s3.connection
access_key = '*'
secret_key = '*'

conn = boto.connect_s3(
    aws_access_key_id = access_key,
    aws_secret_access_key = secret_key,
    host = 'ceph.inoki.cc',
    port = 7480,
    is_secure=False, # uncomment if you are not using ssl
    calling_format = boto.s3.connection.OrdinaryCallingFormat(),
)

bucket = conn.get_bucket('inoki-bucket-2')

key = bucket.new_key('hello.txt')
key.set_contents_from_string('Hello World!')

plans_key = bucket.get_key('hello.txt')
plans_url = plans_key.generate_url(3600, query_auth=True, force_http=True)

print(plans_url)

```

Ce programme sert à créer un objet dans bucket inoki-bucket-2 et générer un lien pour y accéder.

```

(ceph) [inoki@Inoki-Lenovo rgw-test]$ python list-bucket.py
inoki-bucket-1  2019-06-07T15:39:54.193Z
inoki-bucket-2  2019-06-07T15:41:19.291Z
(ceph) [inoki@Inoki-Lenovo rgw-test]$ python create-bucket.py
(ceph) [inoki@Inoki-Lenovo rgw-test]$ python list-bucket.py
inoki-bucket-1  2019-06-07T15:39:54.193Z
inoki-bucket-2  2019-06-07T15:41:19.291Z
inoki-bucket-3  2019-06-07T16:14:46.590Z
(ceph) [inoki@Inoki-Lenovo rgw-test]$ python create-object.py
http://ceph.inoki.cc/inoki-bucket-2/hello.txt?Signature=TNbccWgYKrLnKhyuOUmmLD0Ldew%3D&Exp

```

Les résultats sont ci-dessus.

Tous les composants marchent parfaitement dans notre cluster.

5 Bibliographie

1. CRUSH: Controlled, Scalable, Decentralized Placement of Replicated Data: Sage A. Weil Scott A. Brandt Ethan L. Miller Carlos Maltzahn, Storage Systems Research Center, University of California, Santa Cruz: <https://ceph.com/wp-content/uploads/2016/08/weil-crush-sc06.pdf>
2. Ceph Documentation: <http://docs.ceph.com/docs/mimic/start/intro/>
3. Apache Hadoop:Avantages et Inconvénients: <https://le-datascientist.fr/apache-hadoop-avantages-et-inconvenients?fbclid=IwAR2hRAHkg0kYpDPvcmdmZVzmK-im0TA1E711VueOpmGYqF9nK-mxaEx6oO8>
4. Paxos made Simple, Leslie Lamport Version 01 Nov 2001:<https://www.microsoft.com/en-us/research/uploads/prod/2011/01/paxos-made-simple-Copy.pdf>
5. Paxos Made Live - An Engineering Perspective: Tushar Chandra, Robert Griesemer, Joshua Redstone, Version June 26, 2007: https://static.googleusercontent.com/media/research.google.com/zh-CN//archive/paxos_made_live/IwAR1Z5LOGG4TeaR8rJ4LiK7vNDzroyCy5mt5jXCR3OTR51Hs5fVzSqneJHL0BenchmarkCephClusterPerformance?fbclid=IwAR2VwIPgbU6H
<https://tracker.ceph.com/projects/ceph/wiki/BenchmarkCephClusterPerformance?fbclid=IwAR2VwIPgbU6H>
6. Paxos Wikipedia: [https://en.wikipedia.org/wiki/Paxos\(computer_science\)](https://en.wikipedia.org/wiki/Paxos(computer_science))

Appendices

A Benchmark du Ceph

```
sr05@SR05-1:~/tmp$ dd if=/dev/zero of=here bs=100M count=1 oflag=direct
1+0 records in
1+0 records out
104857600 bytes (105 MB, 100 MiB) copied, 0.454686 s, 231 MB/s
sr05@SR05-1:~/tmp$ dd if=here of=/dev/null bs=100M count=1 iflag=direct
1+0 records in
1+0 records out
104857600 bytes (105 MB, 100 MiB) copied, 0.132493 s, 791 MB/s
```

Figure 23: dd - Écriture et Lecture

```
fyz@fyz-surface4:/mnt/c/Users/en_00$ iperf -c 46.101.122.175
-----
Client connecting to 46.101.122.175, TCP port 5001
TCP window size: 512 KByte (default)
-----
[ 3] local 192.168.43.237 port 58157 connected with 46.101.122.175 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 3]  0.0-10.1 sec  18.9 MBytes  15.8 Mbits/sec
```

Figure 24: iperf - Bande passante du réseau

```

sr05@SR05-1: $ sudo rados bench -p default.rgw.buckets.data 10 rand
hints = 1
  sec  Cur ops   started   finished   avg MB/s   cur MB/s   last lat(s)   avg lat(s)
    0     16      16         0         0         0         -           0
    1     16     120        104       414.657     416       0.420612     0.119993
    2     16     235        219       437.201     460       0.254796     0.136694
    3     16     360        344       458.064     500       0.00279113   0.132904
    4     16     444        428       427.55      336       0.00581746    0.14359
    5     16     518        502       401.234     296       0.00659371    0.152102
    6     16     608        592       394.352     360       0.138362     0.159471
    7     16     701        685       391.147     372       0.0059477     0.158961
    8     16     798        782       390.741     388       0.259253     0.160606
    9     16     894        878       389.983     384       0.00564847    0.161056
   10     16    1012        996       398.171     472       0.00560539    0.158521
Total time run:      10.3865
Total reads made:    1013
Read size:           4194304
Object size:         4194304
Bandwidth (MB/sec):  390.121
Average IOPS:        97
Stddev IOPS:         15
Max IOPS:            125
Min IOPS:            74
Average Latency(s):  0.163216
Max latency(s):      0.720293
Min latency(s):      0.00194385

```

Figure 25: bench - randread, read et write

```

sr05@SR05-1: $ sudo fio testwrite.fio
rbd_iodepth32: (g=0): rw=randwrite, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=rbd, iodepth=32
fio-3.1
Starting 1 process
Jobs: 1 (f=1): [w(1)][100.0%][r=0KiB/s,w=2546KiB/s][r=0,w=636 IOPS][eta 00m:00s]
rbd_iodepth32: (groupid=0, jobs=1): err= 0: pid=292338: Tue Jun 11 19:44:16 2019
write: IOPS=225, BW=902KiB/s (923kB/s)(1024MiB/1162901msec)
   slat (usec): min=2, max=7250, avg=334.24, stdev=510.40
   clat (msec): min=6, max=4452, avg=141.61, stdev=213.94
   lat (msec): min=8, max=4453, avg=141.95, stdev=213.94
   clat percentiles (msec):
    |  1.00th=[ 18],  5.00th=[ 24], 10.00th=[ 28], 20.00th=[ 35],
    | 30.00th=[ 42], 40.00th=[ 52], 50.00th=[ 68], 60.00th=[ 103],
    | 70.00th=[ 159], 80.00th=[ 222], 90.00th=[ 296], 95.00th=[ 439],
    | 99.00th=[ 793], 99.50th=[ 1267], 99.90th=[ 3138], 99.95th=[ 3306],
    | 99.99th=[ 4396]
   bw ( Kib/s): min= 7, max= 2888, per=100.00%, avg=975.08, stdev=480.97, samples=2147
   iops:    min= 1, max= 722, avg=243.71, stdev=120.24, samples=2147
   lat (msec): 10=0.01%, 20=2.15%, 50=36.81%, 100=20.58%, 250=24.82%
   lat (msec): 500=11.74%, 750=2.73%, 1000=0.50%, 2000=0.40%, >=2000=0.25%
   cpu      : usr=45.67%, sys=37.32%, ctx=891120, majf=4, minf=8408
   IO depths : 1=0.1%, 2=0.1%, 4=0.1%, 8=0.1%, 16=0.1%, 32=100.0%, >=64=0.0%
   submit   : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
   complete : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.1%, 64=0.0%, >=64=0.0%
   issued rwts: total=0,262144,0, short=0,0,0, dropped=0,0,0
   latency   : target=0, window=0, percentile=100.00%, depth=32

Run status group 0 (all jobs):
  WRITE: bw=902KiB/s (923kB/s), 902KiB/s-902KiB/s (923kB/s-923kB/s), io=1024MiB (1074MB), run=1162901-1162901msec

Disk stats (read/write):
vda: ios=12473/8440, merge=150/5933, ticks=4848/3780, in_queue=5004, util=0.08%

```

```

sr05@SR05-1:~$ mv testread.conf testread.fio
sr05@SR05-1:~$ sudo fio testread.fio
rbd_iodepth32: (g=0): rw=randread, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=rbd, iodepth=32
fio-3.1
Starting 1 process
Jobs: 1 (f=1): [r(1)][100.0%][r=9676KiB/s,w=0KiB/s][r=2419,w=0 IOPS][eta 00m:00s]
rbd_iodepth32: (groupid=0, jobs=1): err= 0: pid=292180: Tue Jun 11 19:11:38 2019
read: IOPS=1830, BW=7524KiB/s (7704kB/s)(1024MiB/139371msec)
slat (nsec): min=981, max=7375.2k, avg=50914.45, stdev=154598.77
clat (usec): min=4, max=1103.5k, avg=16951.01, stdev=51379.52
lat (usec): min=439, max=1104.3k, avg=17001.92, stdev=51381.98
clat percentiles (usec):
  1.00th=[ 717], 5.00th=[ 1205], 10.00th=[ 2245], 20.00th=[ 4293],
 30.00th=[ 6456], 40.00th=[ 8160], 50.00th=[ 9372], 60.00th=[ 10421],
 70.00th=[ 11207], 80.00th=[ 12387], 90.00th=[ 14746], 95.00th=[ 21365],
 99.00th=[295699], 99.50th=[346031], 99.90th=[641729], 99.95th=[692061],
 99.99th=[952108]
bw ( Kib/s): min= 280, max=11920, per=99.83%, avg=7510.53, stdev=3025.65, samples=278
iops      : min= 70, max= 2980, avg=1877.62, stdev=756.41, samples=278
lat (usec): min=0.01%, 20=0.01%, 50=0.01%, 100=0.01%, 250=0.01%
lat (usec): 500=0.07%, 750=1.16%, 1000=2.27%
lat (msec): 2=5.27%, 4=9.90%, 10=37.19%, 20=38.61%, 50=2.05%
lat (msec): 100=0.56%, 250=1.32%, 500=1.34%, 750=0.21%, 1000=0.02%
lat (msec): 2000=0.01%
cpu        : usr=24.57%, sys=20.07%, ctx=158747, majf=0, minf=7
IO depths  : 1=0.1%, 2=0.1%, 4=0.1%, 8=0.1%, 16=0.1%, 32=100.0%, >=64=0.0%
submit     : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
complete   : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.1%, 64=0.0%, >=64=0.0%
issued rwT: total=262144,0,0, short=0,0,0, dropped=0,0,0
latency    : target=0, window=0, percentile=100.00%, depth=32

Run status group 0 (all jobs):
  READ: bw=7524KiB/s (7704kB/s), 7524KiB/s-7524KiB/s (7704kB/s-7704kB/s), io=1024MiB (1074MB), run=139371-139371msec

Disk stats (read/write):
vda: ios=43/656, merge=0/393, ticks=12/84, in_queue=4, util=0.00%

```

Figure 26: fio - teste R/W de Block device

```

sr05@SR05-1:~$ sudo swift-bench -c 64 -s 4096 -n 1000 -g 100 swift.conf
swift-bench 2019-06-11 19:03:29,964 INFO Auth version: 1.0
swift-bench 2019-06-11 19:03:30,627 INFO Auth version: 1.0
swift-bench 2019-06-11 19:03:32,656 INFO 358 PUTS [0 failures], 177.4/s
swift-bench 2019-06-11 19:03:36,705 INFO 1000 PUTS **FINAL** [0 failures], 164.8/s
swift-bench 2019-06-11 19:03:36,705 INFO Auth version: 1.0
swift-bench 2019-06-11 19:03:37,013 INFO 100 GETS **FINAL** [0 failures], 351.9/s
swift-bench 2019-06-11 19:03:37,013 INFO Auth version: 1.0
swift-bench 2019-06-11 19:03:39,031 INFO 400 DEL [0 failures], 199.5/s
swift-bench 2019-06-11 19:03:42,324 INFO 1000 DEL **FINAL** [0 failures], 188.8/s
swift-bench 2019-06-11 19:03:42,324 INFO Auth version: 1.0

```

Figure 27: swift-bench - teste de RGW