Rapport Final CEPH

Projet SR05

Antonin Deneux, Antoine Protard, Florian Cartier, Paul Jeannot, Pierre-Louis Lacorte, Oshinn Fitzpatrick, Théo Hordequin, Alizée Poirier-Renard

22 Juin 2017

Rapport Final CEPH **0**

I) Présentation **2**

Atouts 3

Inconvénients 3

Concurrents 3

II) Détail du fonctionnement des algorithmes répartis mis en oeuvre dans CEPH: PAXOS et CRUSH **6**

L'algorithme Paxos: 6

L'algorithme de Two-phases commit (2PC) : 6

A quel problème devrait répondre Paxos? 9

Comment Paxos fonctionne? 9

Paxos et les Moniteurs CEPH: 11

L'algorithme CRUSH 12

Répartition des données 12

La CRUSH Map 13

Cluster Map Hiérarchique 13

Buckets 13

Distribution 14

Protection de la surcharge 14

Evaluation de l'algorithme 14

Performance de l'algorithme 16

Fiabilité 18

III) L'architecture CEPH, son installation et son utilisation **19**

Architecture 19

Un problème de maintenabilité 19

RADOS 19

Les « Storages Cluster » 21

IV) Partie pratique: Installation de CEPH sur plusieurs machines **24**

Conclusion **25**

I) Présentation

La tendance depuis plusieurs années, au début des années 2000, est d'émanciper la liaison entre le matériel et le logiciel concernant le stockage de masse. En effet, la problématique liée à la surcharge des systèmes de stockage basés sur du matériel est donc qu'ils ont un impact important dans les budgets informatiques.

CEPH est une plateforme open-source, une technologie de stockage de données distribuées. Elle fait partie de la famille des Software-Defined Storage (SDS). Le principe de cette famille est de dissocier le hardware de l'intelligence qui s'occupe de la gestion des données. C'est un espace de stockage propulsé par un logiciel. Il y a deux types de SDS, ceux qui sont liés à l'hardware qui lui est connecté, et ceux qui en font complète abstraction. CEPH est une alternative aux autres SAN/NAS. Ce principe permet de créer des systèmes de stockage à grande échelle dont les performances et la capacité augmente avec le nombre de nœuds. C'est ce qui est particulièrement apprécié dans le monde des big-data et l'analyse de celles-ci. Bien que cette solution est en plein effervescence, il est important de noter qu'elle n'est pas leader du marché, pour le moment.

C'est la plus ancienne des solutions de SDS en open source. Le logiciel est sponsorisé par Inktank, qui a été racheté par Red Hat en 2014.

Ceph est une couche logiciel qui vient s'ajouter au dessus de la couche hardware. Elle s'adapte à tous types de hardware. Le but étant de pouvoir conserver son hardware et ajouter le logiciel CEPH par dessus, ne nécessitant ainsi pas une entière modification de l'architecture existante. CEPH permet d'accéder à plusieurs disks au travers d'un seul ordinateur.

Il considère les données comme des objets ce qui change des méthodes traditionnelles où les données étaient directement lues depuis les endroits où elles étaient stockées. Il s'appuie sur l'algorithme CRUSH pour calculer l'emplacement des objets et ainsi construire une carte d'ordonnancement des données. Cet algorithme permet au cluster CEPH d'être évolutif, c'est-à-dire de rester viable lorsque le nombre de données croît. La capacité du stockage s'étend à plusieurs pétaoctets (10^15 octets). Il permet également d'équilibrer et de retrouver les données dynamiquement.

CEPH est entièrement géré depuis la ligne de commande. Red Hat propose une interface web appelée Calamari, celle-ci dialogue directement avec le serveur au travers des APIs REST

Atouts

- Agnostique au niveau matériel. Cela permet de construire des clusters hétérogènes sans être contraints de se fixer à une marque en particulier et ainsi réduit les coûts de stockage.
- Interface RESTful permet un accès aux API telles que S3 d'Amazon et Swift.
- Scalable et tolérant aux pannes (via la distribution ou erasure-coding)
- Équipés pour les snapshots
- L'activité du projet, les améliorations sont nombreuses et régressions corrigés rapidement

Inconvénients

- Pas supporté par les clients Windows
- Vu l'activité du projet, faire attention à utiliser les versions stables, et mettre à jour relativement souvent.

Concurrents

Il existe d'autres systèmes de stockage que les solutions SDS, plus traditionnelles:

Le SAN (Storage Area Network) permet un accès bas niveau au disque de stockage par le réseau du centre de stockage. Les clients sont reliés par un réseau très haut débit à des équipements d'interconnexions afin d'interagir aux disques de stockages. Le logiciel est lié aux systèmes de fichiers.

Le système NAS (Network Attached Storage) est un dispositif de stockage en réseau. Il s'agit d'un serveur de stockage à part entière pouvant être facilement attaché au réseau de l'entreprise afin de servir de serveur de fichiers et fournir un espace de stockage tolérant aux pannes. Il embarque un système d'exploitation et un logiciel qui permet de le configurer. Il possède son propre système de fichier. Le stockage NAS désigne un produit spécifique qui se situe à mi-chemin entre le serveur d'applications et

le système de fichiers, tandis que le réseau SAN fait référence à l'architecture. Ce dernier est son propre réseau, puisqu'il connecte l'ensemble des unités de stockage et des serveurs.

CEPH a des concurrents dans divers domaines, d'une part par son système de fichier (CephFS) et d'autres part son système de stockage (SDS).

L'entreprise Red Hat propose une autre solution de stockage de fichiers. Il n'utilise cependant pas la même trame que pour le CEPH. C'est le Red Hat Gluster Storage.

Red Hat Gluster Storage est basé sur le système de fichier GlusterFS (présenté à la partie suivante). La solution est particulièrement efficace lorsqu'il s'agit de gérer la croissance des données non structurées de l'entreprise pour les charges de travail comme les gros volumes de données et les analyses, la virtualisation d'entreprise et l'hyper convergence, ainsi que le stockage des contenus multimédias et d'archives.

Ensuite, nous allons présenter un autre système de fichiers distribués qui peut être utilisé par une entreprise à la place CEPH.

GlusterFS est un système de fichiers de clusters reposant sur le modèle client-serveur, il y a donc un ou plusieurs Server Gluster et pareil pour Client Gluster. Les serveurs sont typiquement déployés comme des «briques de stockage», chaque serveur exécutant un démon « glusterfsd » qui exporte un système de fichier local comme un « volume ». La plupart des fonctionnalités de GlusterFS sont implémentées comme « translators / traducteurs », incluant : la duplication et la réplication par fichier, le partage de charge par fichier, la gestion des pannes, l'ordonnancement et le cache disque. Par défaut, les fichiers sont simplement distribués entre les différents serveurs et sont protégés par des backups réguliers. Pour contrer les points de défaillance, GlusterFS s'assure de ne pas avoir de point central de défaillance. Le mode de lecture/écriture le plus courant est le mode distribué et répliqué.

On peut noter qu'il y a aussi MooseFS ou Lustre par exemple.

Il existe d'autres SDS, en voici trois :

Scality est une entreprise développant sa propre solution SDS. Elle est basé en Californie. Leur produit est le Scality RING. Ce logiciel permet une intégration S3. Il

prend en charge Microsoft Active Directory(AD) et de Amazon Web Services (AWS). Il possède aussi des API HTTP REST.

Scale-IO (Dell) est une technologie SDS en mode bloc qui permet de créer des pools de stockage mutualisés en agrégeant la capacité disque présente sur les serveurs. Elle est disponible en téléchargement libre. Les noeuds peuvent être intégrés en OpenStack (Cinder), mais aussi d'autres tels que VMWare ou Mirantis. Elle supporte l'IPv6 et LDAP. Les données passent par un checksum à la source afin de vérifier la non corruption de celles-ci. ScaleIO supporte jusqu'à 5 nœuds pour la gestion des métadonnées du cluster. La défaillance engrange une nouvelle répartition des noeuds de métadonnées. L'interface peut passer par OpenStack, Docker ou encore VMWare Photon OS (Ubuntu est supporté). ScaleIO dispose aussi d'une API RESTful pour accéder aux clusters.

CleverSafe DSNet. Ils ont introduit l'approche erasure-coding qui est une méthode de stockage sur disques, alternative à RAID. En 2015, il a été racheté par IBM.

On peut en dénombrer beaucoup plus en ouvrant les horizons des autres types de stockages (NAS/SAN ou encore les solutions liant matériel/logiciel qui sont encore beaucoup utilisés). Le choix est multiple et est laissé à l'appréciation des administrateurs du système de l'entreprise le plus souvent, selon les fichiers manipulés et la topologie des réseaux de l'entreprise.

II) Détail du fonctionnement des algorithmes répartis mis en oeuvre dans CEPH: PAXOS et CRUSH

L'algorithme Paxos:

Dans le cadre de la programmation distribuée, il est forcément intéressant de parler de consensus entre processus.

CEPH n'échappe pas à la règle et implémente l'algorithme PAXOS pour établir des consensus entre tous ces services. Les services de CEPH qui sont sujet à cet algorithmes sont :

- map des OSD
- map des Moniteurs
- map des PG
- map de CRUSH

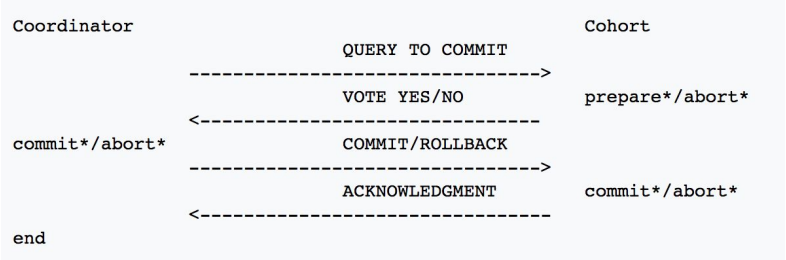
Avant de parler de PAXOS il est intéressant de voir à quoi cet algorithme répond. Pour ce faire nous parlerons des algorithmes sur lesquels il se base.

L'algorithme de Two-phases commit (2PC) :

L'algorithme 2PC est un algorithme qui permet de réaliser des transactions dans un système distribué de telle sorte que chaque site se mette d'accord sur les valeurs de leurs variables. Il consiste en deux phases principales :

1. Phase de vote : Un coordinateur contacte tous les noeuds et leur **suggère** une valeur finale à modifier. Les noeuds lui disent s'ils sont d'accord ou non.

2. Si tout le monde est d'accord alors le coordinateur contacte tous les noeuds pour leur dire que la valeur est finale. Si l'un des noeuds n'est pas d'accord alors il informe tous les noeuds que la valeur n'est pas finale.



Les votes ne sont que sur la valeur proposée. Un noeud ne peut répondre que par oui ou non. Il ne peut pas suggérer son alternative. Si un noeud veut suggérer sa valeur il doit lui même passer par ces deux phases. Cependant l'algorithme n'est pas très efficace puisque pour N noeuds il y a 3N messages envoyés.

Que se passe t-il si un noeud crash alors qu'il était en phase 1?

Quand un noeud démarre un 2PC alors qu'un autre noeud a démarré un 2PC; alors il ne recevra pas de vote oui/non puisque les noeuds ayant démarrés un 2PC sont bloqués jusqu'à la prochaine phase. Ce problème peut être résolu avec d'autres noeuds qui gèrent la coordination en observant les timeout.

Que se passe t-il si un noeud crash alors qu'il était en phase 2?

Le noeud peut entrer en contact avec tous les autres noeuds pour découvrir leurs votes (cela implique que les noeuds se souviennent de leur votes) et mener à bien la transaction en la finissant. Problème: d'autres crashs peuvent avoir lieu ce qui entraînerait alors une transaction qui ne prendrait jamais fin. 2PC ne peut donc pas fonctionner de manière sécurisée sur les crash des noeuds.

Algorithme Three-phases commit (3PC) :

Le problème soulevé par le 2PC est que dans le cas où le coordinateur crash alors le protocole peut très bien ne jamais se terminer. Ce problème peut être résolu en ajoutant une phase :

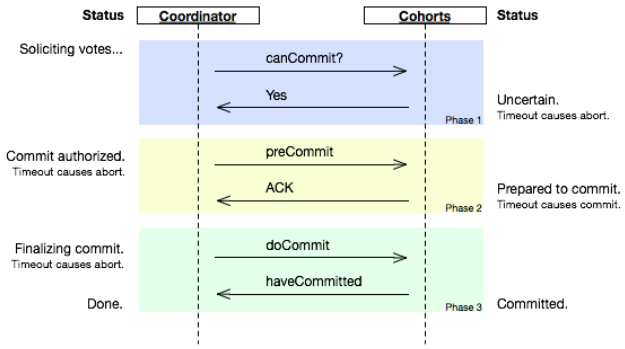
1. Phase de vote : Un coordinateur contacts tous les noeuds et leur **suggère** une valeur finale à modifier. Les noeuds lui disent s'ils sont d'accord ou non.

2. Quand le coordinateur reçoit un yes de tous les noeuds alors il leur envoit un message "*prepare to commit*". L'objectif de ce message est que les noeuds "enfants" puissent faire un undo de leur action.

3. Si le coordinateur reçoit un message d'acknowledge de tous les noeuds alors il peut communiquer le résultat du vote. Dans l'autre cas, si tous les noeuds ne valident pas la transaction alors ils peut prévenir d'un undo. Finalement dans tous les cas, si le coordinateur crash alors n'importe quel autre noeud peut reprendre son rôle.

- Si n'importe quel noeud prévient le coordinateur qu'il n'a pas reçu de "*prepare to commit*" message alors on peut abandonner l'action.

- Si un noeud qui a commit la transaction crash, on sait que tous les autres noeuds on reçu aussi une autorisation au "*prepare to commit*" donc on peut quand même effectuer la transaction.



Ainsi, l'ajout d'une troisième phase protège d'un crash de noeud mais entraîne donc un ralentissement du système puisque c'est le coût d'un ajout de message entre N noeuds.

Le point sur lequel 3PC est limité est celui de la partition du réseau.

A quel problème devrait répondre Paxos?

Le problème qui semble ne pas être résolu est celui de la partition de la sauvegarde en réseau. L'autre problème auquel devrait répondre Paxos est qu'on a traité ici est le cas des crashs des noeuds. Que se passe t-il si un noeud crash et redémarre par la suite? Comment récupérer l'information?

Comment Paxos fonctionne?

Pour l'exécution de l'algorithme PAXOS il faut que les sites puissent supporter 5 rôles : client (fait la requête), Acceptors, Proposer, Learner et Leader (un Proposer particulier)

1. Sélection d'un noeud Leader/Proposer

2. Le leader sélectionne une valeur et envoie cette dernière a tous les autres noeuds (Acceptors) à travers un accept-request message. Les noeuds Acceptors peuvent rejeter ou accepter ce message

3. Quand une majorité des noeuds ont acceptés alors un consensus est atteint et le noeud Leader peut envoyer un message de commit à tous les autres noeuds.

Paxos ne nécessitant que la moitié des noeuds pour valider un consensus il est résistant même dans le cas ou la N/2 - 1 des noeuds ne répondent pas.

Pour supporter le cas ou c'est le Leader qui meurt, Paxos autorise plusieurs noeuds qui essaient de prendre le Leadership. Cela implique qu'à un instant donné il peut y avoir deux Leaders qui proposent des valeurs différentes. Pour obtenir un consensus Paxos met en place deux mécanismes:

1. Assigner un ordre aux Leaders

2. Quand un consensus est atteint sur une valeur, Paxos va forcer les futurs Leaders à avoir la même valeur afin d'assurer un suivi du consensus. Ceci est rendu possible par le fait que les acceptors envoient la valeur la plus récente pour laquelle ils ont donné une autorisation.

Finalement l'algorithme de Paxos est constitué de 2 phases mais trouve des mécanismes pour éviter les échecs de noeuds.

Phase 1 :

- Un noeud choisi de devenir Leader
    - Choisi une séquence x et une valeur v et envoi la proposition P1(x,v) à tous les acceptors
- Quand un acceptor reçoit une proposition P1(x,v)

    - Si c'est la première proposition qu'il reçoit alors il lui renvoie "oui". Toutes les propositions inférieure à x seront rejetées
    - Si ce n'est pas la première proposition qu'il reçoit :
        - Compare x avec la séquence la plus élevée qu'elle a acceptée (mettons qu'elle a accepté la proposition P2(y,v2)
        - si x < y alors rejet de la proposition P1 avec y comme séquence
        - si x > y alors acquittement avec P2(y,v2)

Phase 2 :

- Si une majorité des acceptors ne répondent pas, ou répondent avec un rejet alors le leader abandonne sa proposition et recommencera peut être plus tard

- Si la majorité des acceptors répondent par un accord alors le leader va recevoir les valeurs que les acceptors ont accepté avant sa proposition. Il va choisir une de ces valeurs (si aucune valeur n'a été choisi alors il prends la sienne) et envoyer un message "accept" avec P(x,"valeur choisie")

- Quand un Acceptor reçoit un message "accept"
    - si la valeur est la même que toutes les propositions acceptées et que la séquence est la plus haute que l'acceptor ait acceptée alors -> "accepted"
    - sinon -> "reject"

- Si le leader ne reçoit pas un message "accepted" de la majorité alors il abandonne la transaction et la reprendra peut être plus tard. Si au contraire il reçoit une majorité alors le protocole peut être considéré comme terminé. Comme optimisation on pourrait ajouter le fait que le Leader envoie un message de commit aux acceptors mais au prix d'un ralentissement des transactions.

Paxos et les Moniteurs CEPH:

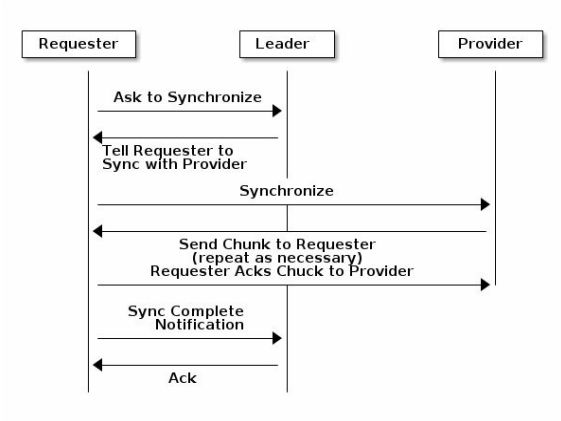
Dans le cadre de CEPH, l'algorithme PAXOS permet de maintenir une cohérence notamment sur la map des moniteurs. Les moniteurs doivent trouver un consensus sur chaque ajout ou update de moniteur. La séquence x de l'algorithme paxos dans les messages est incrémental et permet donc de maintenir un historique des moniteurs. En effet si un moniteur s'éteint et par la suite se rallume il pourra ainsi rattraper son "retard".

Quand CEPH est en production sur un cluster comprenant différents moniteurs alors chaque moniteur vérifie s'il a la même map de moniteur que leurs voisins. Il arrive aussi que parfois un moniteur crash et qu'il se rallume par la suite. Il doit alors se re-synchroniser. Une synchronisation se réalise de la manière suivante :

Le premier moniteur à finir l'algorithme PAXOS devient Leader.

Un autre moniteur qui a aussi la dernière version de la map devient 'provider'. C'est lui qui sera chargé de mettre à jour le moniteur qui a besoin de synchronisation. Un autre noeud est chargé de la synchronisation pour éviter de surcharger le Leader.

Enfin 'requester' est le noeud qui doit être synchronisé.



L’algorithme CRUSH

CRUSH, qui signifie /Controlled, Scalable, Decentralized Placement of Replicated Data/ est un algorithme scalable et pseudo-aléatoire de répartition de données sur des périphériques hétérogènes. Celui-ci permet de déterminer où et comment, stocker et retrouver des données distribuées sur un ensemble d’espace de stockage. Il permet également de gérer de manière optimisée l’ajout et la suppression d’espaces de stockage en minimisant les déplacements de données. Enfin, il est entièrement configurable selon les installations des utilisateurs et leurs volontés en terme de sécurité et distribution des données (séparation des réplicas sur différents espaces indépendants en cas de panne).

Répartition des données

Plutôt que d’utiliser un fichier d’index regroupant l’ensemble des localisations des différents fichiers, CRUSH utilise un algorithme de hashage permettant de générer l’adresse de stockage à partir du nom du fichier, ainsi que de la CRUSH Map. Cette méthode permet d’éviter de tout centraliser dans un fichier et ainsi éviter que l’ensemble du système ne dépende que d’un seul fichier : point individuel de défaillance (single point of failure). En terme de performance, cela évite l’apparition d’un goulot d’étranglement au niveau du fichier d’index, et supprime également la limite physique liée au fichier lui même en terme de scalabilité.

CRUSH répartit de manière uniforme les fichiers à stocker sur l’ensemble des espaces de stockage disponibles. En effet, celui-ci répartit aléatoirement les données entre les différents espaces et entre les données récentes ou plus anciennes. L'ajout d’un nouvel espace de stockage pourrait déstabiliser l’équilibre fait entre tous les espaces: les nouveaux fichiers et les plus anciens. Afin d’éviter tout déséquilibre, un échantillon de données est déplacé vers le nouveau périphérique afin de garder l’équilibre.

Grâce à cette méthode, l’ensemble des périphériques seront toujours chargés de manière équitable, permettant au système d’être performant qu’importe la charge de travail demandée.

CRUSH est implémenté comme une fonction pseudo-aléatoire mais déterministe, prenant en paramètre un objet (agissant comme un identifiant), et donnant en sortie une liste de périphériques sur lesquels stocker les copies du fichier source.

La CRUSH Map

CRUSH doit posséder une carte de l'installation, appelée CRUSH Map qui décrit l'architecture et l'organisation du réseau. Cette topologie doit être hiérarchique: souvent effectuée sous forme d'arbre, ayant comme feuille un OSD. Ce fichier explicite également les règles de répartition des données de manière à régler très précisément les contraintes de duplication. Ainsi, ce fichier précise l'ensemble des périphériques disponibles (ex: 3 disques durs sur un rack, 5 SSD sur une baie dans un autre bâtiment) ainsi que les contraintes de duplication (ex: chaque fichier doit être copié, et les copies doivent être dans des baies de stockage différentes, qui ne partagent pas la même alimentation électrique).

Cette approche a deux avantages clés :

1. Système entièrement distribué : chaque client peut, de manière totalement indépendante, calculer la localisation d'un objet à partir de son identifiant et de la topologie du réseau.
2. Le fichier CRUSH Map est presque statique, puisqu'il n'est modifié que lorsqu'un périphérique de stockage est ajouté ou supprimé.

Ainsi, la CRUSH Map peut être divisée en 2 parties distinctes :

1. Cluster map
2. Placement rules

Cluster Map Hiérarchique

La cluster map est composée d'un ensemble de 'devices' et 'buckets'. Chacun d'eux a un identifiant (numérique) et un poids associé.

Buckets

Un bucket est un conteneur regroupant un ensemble de devices (c'est donc un panier contenant un certains nombre d'appareils). Cette relation peut être comparée à la relation père-fils d'un arbre, et la cluster map peut donc être représentée comme un arbre (d'où la notion de hiérarchie).

Les feuilles de l'arbre sont des devices, et leurs pères sont des buckets. Les poids associés à ces deux entités sont fixés par l'administrateur. Il représente la quantité de données dont ils sont responsable pour le stockage.

Il est important que la représentation soit claire et précise de manière à faciliter la maintenance et la sécurité de l'application en cas de modification de la topologie.

Le poids d'un bucket correspond au poids des fils accumulés, donc à la somme des buckets et devices qui lui sont rattachés.

Le placement de données dans la hiérarchie est effectuée en sélectionnant récursivement les buckets.

Uniform Bucket:

Tous les devices du bucket sont identiques. Il est donc très rapide pour répartir les données entre les buckets (O(1)). Cependant, si un device est ajouté/supprimé, il faut tout redistribuer car la répartition n'est plus la même. L'efficacité est mauvaise lorsqu'il faut modifier la distribution des copies.

Distribution

La distribution des données par CRUSH apparaît aléatoire, et non corrélée avec l'identifiant de l'objet ou les OSD ciblés. Ainsi, empiriquement, elle suit une loi binomiale qui peut être approximée par une gaussienne lorsque le nombre d'objets est grand.

Protection de la surcharge

Afin de se protéger contre la surcharge d'un composant, CRUSH possède un mécanisme de correction permettant de redistribuer un pourcentage des données stockées d'un appareil à la limite de sa capacité.

De plus, lorsqu'un composant semble être peu performant, CRUSH le traite comme un appareil "Partiellement défaillant" ce qui active le mécanisme ci-dessus et corrige les défauts dus aux grandes charges de travail.

Evaluation de l'algorithme

De nombreux essais ont été effectués afin de mesurer empiriquement la complexité et l'efficacité de l'algorithme CRUSH par rapport à RUSHT et RUSHP. Ces

expériences ont été effectuées sur un modèle d'une profondeur de 4 étages. Ainsi, le réseau était composé de 7290 appareils

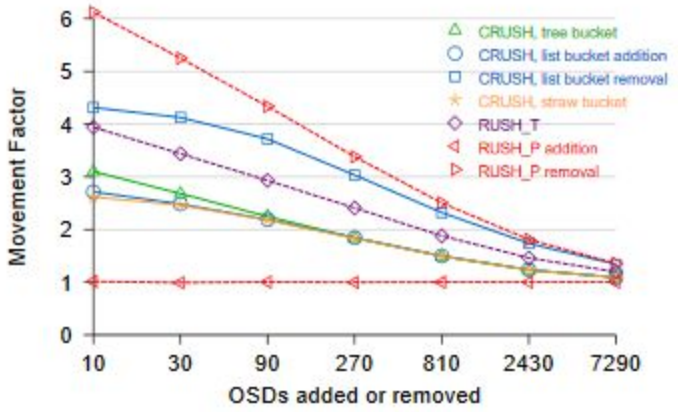


Figure 1

Pour la figure ci-dessus, l'axe X représente le nombre d'OSD ajoutés ou enlevés et l'axe Y, le facteur de déplacement qui équivaut au nombre d'opérations effectués sur le nombre optimal d'opérations.

On remarque facilement que plus la taille de changement à faire est grande, plus la réorganisation sera efficace quel que soit l'algorithme utilisé. De plus, il s'avère que CRUSH obtient des résultats moyens indépendamment de l'opération réalisée, ce qui n'est pas le cas de RUSHP.

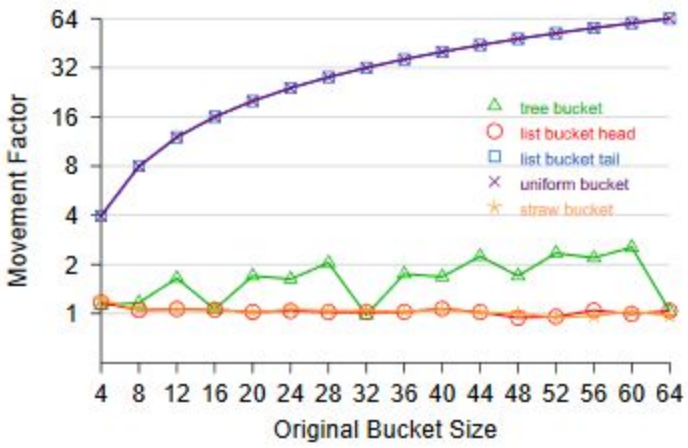


Figure 2

La figure 2 montre l'évolution du facteur de mouvement en fonction de la taille du Bucket et de son type. Ici, on remarque que l'utilisation d'un uniform Bucket engendre un réarrangement total ce que ne peut être toléré dans un système efficace. On constate également qu'un système hybride permettrait de conserver une bonne cartographie du réseau tout en évitant les pires scénarios de chacune des solutions (ex: retirer le dernier objet d'une liste).

Performance de l'algorithme

Pour un réseau de n OSDs, calculer la CRUSH MAP est de l'ordre O(log(n)) ce qui permet d'évaluer rapidement la localisation d'un objet dans le cluster.

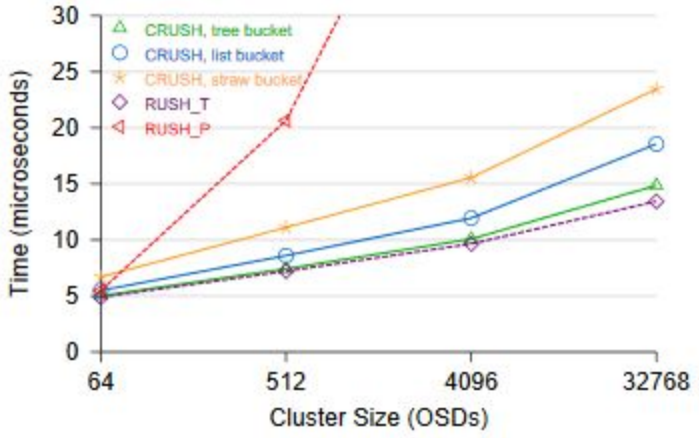


Figure 3

La figure 3 présente l'évolution du temps de calcul de la CRUSH map en fonction de la taille du réseau et de l'algorithme utilisé. On remarque que seul l'algorithme RUSHT obtient de meilleurs résultats que CRUSH, dû à une complexité légèrement inférieure.

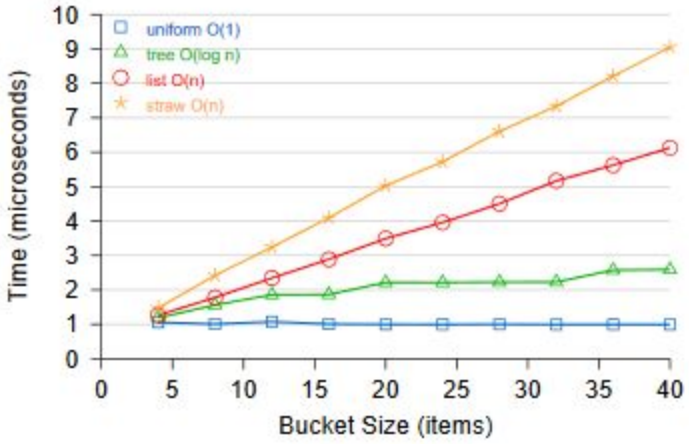


Figure 4

Enfin, la figure 4 présente le temps nécessaire pour recherche une copie d'un objet dans le cluster en fonction du type et de la taille des Buckets. On constate que le

tree bucket permet d'obtenir une complexité en log(n) tandis que le temps pour les list et straw est linéaire en fonction de la taille du bucket.

Enfin, il faut savoir que 45% du temps d'exécution de l'algorithme CRUSH est effectué au niveau de la fonction de hashage (Jenkin's 32-bithashmix). Ainsi, la plupart des optimisations doivent être avant tout portées sur l'amélioration de cette fonction, afin d'assurer une distribution rapide et de qualité.

Fiabilité

La fiabilité est d'une importance critique pour l'algorithme CRUSH. Par exemple, pour un réseau de 1000 appareils, même dans le cas critique où 50% du réseau est marqué comme défectueux, le calcul de la CRUSH MAP augmente seulement de 71%, ce qui permet de maintenir un niveau de service convenable.

De plus, la topologie du réseau en peer-2-peer engendre deux effets opposés. En effet, le fait que seul peu de bits de données répliquées soient sur un même appareil, permette un rétablissement plus rapide en cas d'échec. Cependant, un grand réseau implique souvent une augmentation de la probabilité de perdre une partie des données partagées. C'est pourquoi la CRUSH MAP contient des règles permettant de définir quels sont les appareils alimentés par les mêmes sources, ou placés dans les mêmes armoires. Ainsi, cela prévient le risque de perdre des données en cas d'erreur majeure sur un ensemble de composants.

III) L'architecture CEPH, son installation et son utilisation

Architecture

Un problème de maintenabilité

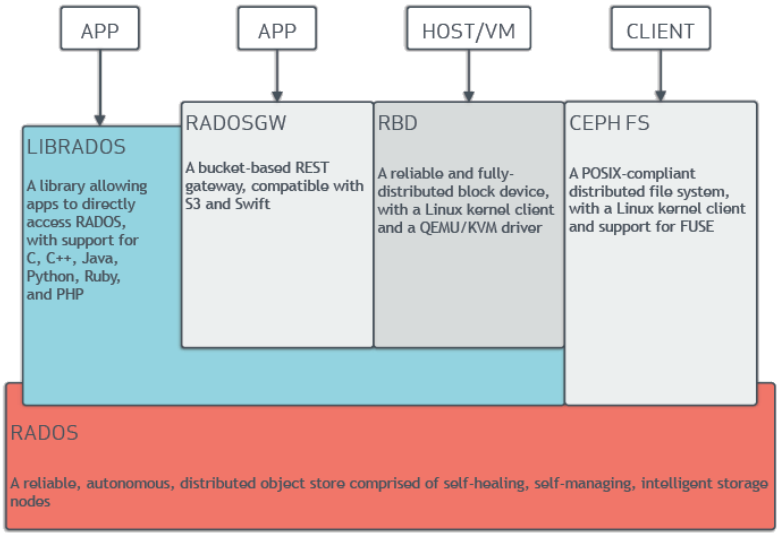
Les architectures de stockages à base d'objets (object storage device, **OSD**s) sont apparues dans le but de rendre les clusters de stockage plus maintenable. Cependant les systèmes existants continuent de considérer les noeuds de stockage comme des dispositifs passifs, malgré le fait qu'ils présentent une certaine capacité d'intelligence et d'autonomie.

Les architectures émergentes de stockage de clusters aspirent à exploiter un peu plus l'intelligence des OSDs en leur confiant le choix de l'allocation des données afin de simplifier l'accès aux données à partir d'un client. Cependant, selon les ingénieurs d'InkTank, les systèmes utilisant cette architecture n'exploitent pas encore assez les possibilités des ODS, car ceux-ci ne font toujours quasiment que répondre aux requêtes de lecture et d'écriture.

Quand ces systèmes de stockages comprennent plusieurs milliers de noeuds, la gestion constante du placement des données, de la détection de pannes et de la récupération des données par les clients deviennent un poids pour ces derniers.

RADOS

RADOS (Reliable, Autonomic Distributed Object Store) a été implémenté afin d'exploiter au maximum l'intelligence des OSDs dans le but de distribuer la complexité liée à l'accès persistant aux données, le stockage redondant, la détection d'erreurs et la récupération de données perdus dans un système contenant plusieurs milliers d'appareils de stockage. Il donne de meilleurs performances que les systèmes stockages précédent tout en restant maintenable et en donnant l'illusion aux clients d'un système de stockage objet local.



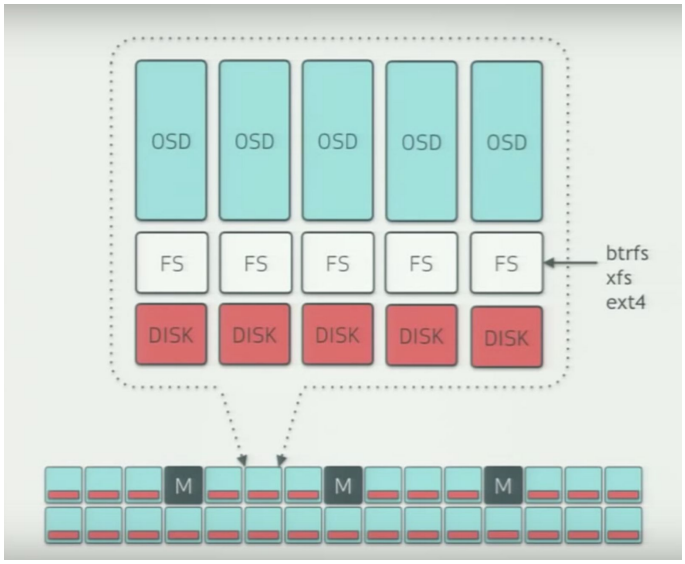
Comme le montre la figure ci-dessus, il y a quatre modes de connexions a RADOS :

- Au travers d'applications, des librairies permettant d'accéder directement aux OSDs sont proposées pour la plupart des langages de programmation communs comme C, C++, Java, Python, Ruby ou PHP.
- Un mode de stockage objet : via la passerelle RADOSGW, CEPH propose un mécanisme de stockage objet de type "bucket" accessible via des API compatibles avec Amazon S3 et OpenStack Swift.
- Un mode de stockage en mode bloc : via le pilote noyau RBD, un hôte Linux peut monter un volume CPEH et l'utiliser comme un disque local. Des intégrations sont aussi fournies avec KVM/QEMU pour fournir du stockage en mode bloc à des machines virtuelles.
- Un stockage en mode fichier : via CephFS, CPEH propose un accès en mode fichiers compatible POSIX et intégré avec OpenStack Manila. CephFS est considéré comme stable depuis la version "Jewel" de CEPH. La mise en œuvre de CephFS nécessite l'installation de serveurs de métadonnées spécifiques en plus des serveurs habituellement déployés pour un cluster Ceph.

Quand la quantité de données à stocker est de l'ordre du petabyte, les systèmes sont forcément dynamiques. Ils sont construits progressivement et évolue en suivant l'ajout de nouveaux noeuds ou le retrait des plus anciens, certains subissent des pannes

et de nombreuses données sont créées et détruites. RADOS assure un vue cohérente de la distribution des données et un accès cohérent en lecture écriture aux OSDs grâce à l'utilisation d'un Cluster Map comportant des numéros de versions.

Les « Storages Cluster »



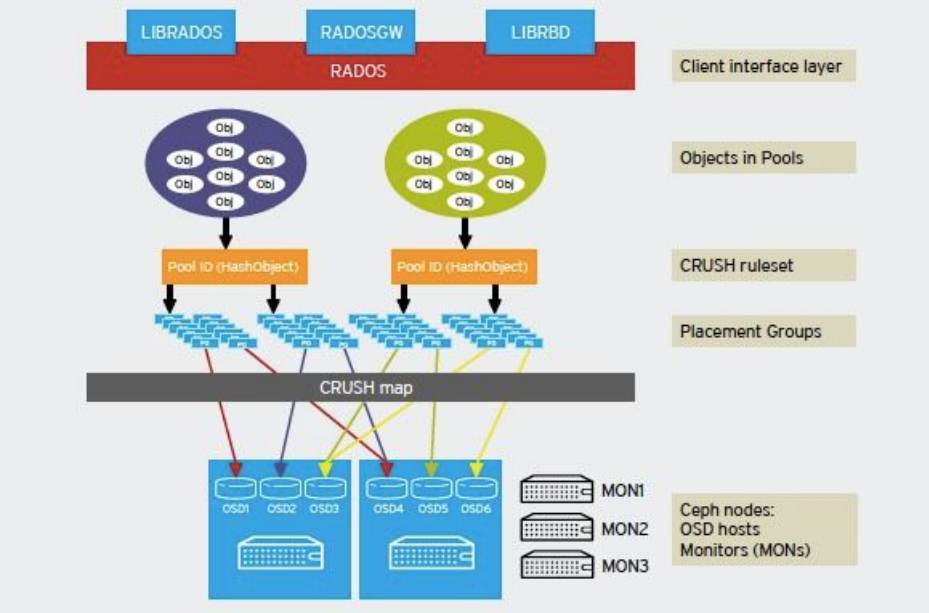
CEPH propose deux cluster de stockage, les **OSDs Deamons** et les **Monitors**.

La figure ci-contre détaille le contenu d'un OSD Daemon. Celui-ci contient donc un ensemble de disques durs, ainsi qu'un système de fichier et un OSD pour chacun d'entre eux.

Les monitors gardent une copie de la disposition du cluster. Un cluster de monitors permet de toujours assurer la disponibilité des données même en cas de panne d'un OSD Daemon. Les OSDs clients reçoivent une copie de la cluster map à partir des monitors.

Les OSDs Daemons agissent comme des agents de contrôle, ils vérifient leur état et celui des autres OSDs et le rapportent aux Monitors. Ils utilisent, tout comme les clusters clients, l'algorithme CRUSH pour gérer la distribution des données, au lieu dépendre constamment d'une table de consultation centrale.

Les OSDs Daemons sont en charge de la réplication et de la récupération des données tandis que les monitors établissent un consensus lors des phases de décisions. C'est pour cette raison qu'il y a peu de monitors dans les clusters, il serait difficilement gérable d'avoir un trop grand nombre de noeuds prenant les décisions.



Un cluster Ceph stocke les données sous forme d'objets stockés dans des partitions logiques baptisées "pools". À chaque pool Ceph correspond un ensemble de propriétés définissant les règles de réplications (le nombre de copie pour chaque donnée inscrite) ou le nombre de groupes de placement (placement groups) dans le pool. Les pools peuvent être répliqués ou protégés par l'utilisation de code à effacement (erasure coding). Dans un pool répliqué, le système stocke autant de copies que spécifié de chaque donnée (le coût de stockage augmente linéairement avec le nombre de copies spécifié). Par exemple, si l'on a spécifié trois copies et que le cluster dispose de trois nœuds, la triple réplication permet de survivre à deux pannes de nœuds ou à la panne de deux disques. Dans un pool Ceph avec erasure coding, chaque objet écrit sur le cluster est découpé en n segments (avec n=k+m, ou k est le nombre de segments de données et m le nombre de segments contenant des données de parité). Il est possible de spécifier le domaine de faille (par exemple un rack) de telle sorte que par exemple, dans un schéma ou n=3+2, les données seront réparties sur des disques situés dans cinq racks différents. À l'extrême, les données peuvent ainsi survivre à deux pannes complètes de rack. Il est à noter que Ceph propose aussi un mécanisme de snapshot de pool.

Un groupe de placement est un conteneur logique utilisé par Ceph pour segmenter la distribution des objets entre OSD. Les groupes de placements sont en fait des fragments de pools assignés à des OSD spécifiques. L'idée est en particulier de créer un niveau d'indirection supplémentaire afin de faciliter l'équilibrage des données dans les clusters, mais aussi de faciliter les reconstructions de données en cas de panne d'un

composant. Lors de la création d'un pool, il est impératif de fournir le nombre de groupes de placement que l'on souhaite mettre en œuvre. L'algorithme de distribution de données de Ceph, baptisé Crush, utilise en effet cette valeur pour distribuer les données.

IV) Partie pratique: Installation de CEPH sur plusieurs machines

Installation sous CentOs

Un des avantages de CEPH est qu'il est basé sur des communications TCP/IP et qu'il n'a pas besoin de matériel dédié spécifique. Cependant, les différents composants d'un cluster CEPH ont des besoins différents en termes d'espace disque et de capacités de calculs.

Les recommandations en termes de hardware de CEPH sont les suivantes :

- Les OSD ont besoin de beaucoup d'espace de stockage, qu'il soit de la forme d'un RAID ou d'un disque dur simple. L'utilisation de RAID est redondante car la perte de donnée est gérée par CEPH a un niveau supérieur. Les OSD font tourner le service RADOS et utilisent l'algorithme CRUSH, ils ont donc besoin de capacités de calculs suffisantes.
- Les moniteurs n'ont pas besoin d'être très puissants ni d'avoir de grande capacités de stockages.
- Les serveurs MD ou metadata ont besoin de fournir leurs informations rapidement donc une bonne quantité de RAM est nécessaire.

Dans un environnement de production CEPH conseille de séparer les OSD avec un disque dédié à l'OS et un ou plusieurs disques dédiés aux stockages des données. CEPH recommande pour ces disques d'utiliser le file system btrfs. CEPH utilise XATTR du système fichier sous-jacent pour certaines tâches comme les snapshots metadata ou les access list dédiées à RADOS. Ext4 possède une limite de 4KB pour XATTR, ce qui peut causer une interruption de service d'un OSD lors d'un calcul de snapshot.

Dans notre cas nous allons utiliser la même configuration physique pour les OSD et les moniteurs, ainsi que les nœuds admin et client ; ayant rencontré des difficultés lors du déploiement sous Ubuntu nous nous sommes tournés vers l'OS CentOs pour nos moniteurs, OSD, et le noeud admin.

Avant de télécharger les paquets CEPH, la première étape est d'obtenir la clé pgp permettant de communiquer de manière sécurisée avec le serveur de CEPH.

Pour cela il faut faire un wget sur la clé et l'ajouter à notre répertoire de clés. Pour que le yum install marche sur nos machines il a fallu ajouter le proxy UTC dans la configuration de yum.

```
sudo rpm -Uhv http://download.ceph.com/rpm-jewel/el7/noarch/ceph-release-1-1.el7.noarch.rpm
```

Ensuite nous pouvons ajouter le package CEPH souhaité dans notre liste de package.

Ensuite un simple sudo yum update suivi d'un yum install ceph permet d'installer ceph sur la machine courante. La documentation de CEPH conseille d'utiliser chef pour gérer le cluster, mais nous l'avons fait manuellement.

Nous avons donc ajouté les clés SSH de chacun des nœuds à notre nœud administrateurs pour permettre à celui-ci de se connecter à elles et faire les modifications nécessaires lors de la configuration du cluster.

Configuration

CEPH utilise lors du lancement des daemons des fichiers de configurations *ceph.conf* présents sur chaque hôtes. Dans ces fichiers de conf nous pouvons interagir avec de nombreux paramètres de CEPH qui permettent de configurer le comportement de notre cluster.

Ces paramètres peuvent aussi être modifiés pendant que le service tourne en utilisant des lignes de commandes.

Le fichier de configuration permet de définir :

- L'appartenance au cluster
- Le nom des hôtes
- Les chemins jusqu'aux hôtes
- Les paramètres du service

Les fichiers de configurations fonctionnent d'une manière similaire au feuilles CSS : une définition à un niveau inférieur remplace une définition à un niveau supérieur.

Le niveau [global] est le niveau le plus élevé : il s'applique à tous les nœuds. Les paramètres définis au niveau global peuvent être redéfinis pour un groupe d'hôtes particuliers, par exemple [osd] permet de définir les paramètres communs à tous les OSD, comme [mon] pour les nœuds moniteurs etc.

De plus certains paramètres sont spécifiques à certains types de nœuds, il y a donc des paramètres spécifiques modifiables pour les OSD, ainsi que des paramètres spécifiques modifiables pour les moniteurs, ou les serveurs metadata.

Quelques exemples de ces paramètres sont :

- osd max write : la taille maximum pouvant être écrite
- osd client message size : taille maximum de message pouvant venir d'un client
- osd stat refresh : l'intervalle de rafraîchissement du statut de l'OSD
- osd min down reporters : nombre de fois qu'un OSD doit rapporter qu'un autre OSD est down
- mon osd nearfull ratio : %d'occupation d'un OSD qui le fait être considéré « full »
- mon tick interval
- mon osd down interval
- mds log skip corrupt events
- ...

Tous ces paramètres possèdent des valeurs par défauts qui peuvent ne pas convenir pour un cluster spécifique. Il s'agira de modifier ces valeurs pour qu'elles correspondent à la configuration que l'on cherche, selon le matériel que nous avons utilisé dans notre cluster, sa taille, et l'utilisation que nous allons en faire.

Il est possible aussi de modifier certains paramètres pour une instance précise, pour cela il faudra indiquer le nom de l'hôte, [osd.1] par exemple.

Les hôtes font parties de sous ensemble du cluster qui s'appellent des pools. Lors de notre projet nous avons utilisé les pool par défauts de ceph qui sont data, metadata, et rbd.

Voici notre fichier /etc/hosts définissant un DNS local :

```
Host ceph-admin
  Hostname grapcli09
  User cephuser
Host mon1
  Hostname grapcli11
  User cephuser
Host osd1
  Hostname grapcli12
  User cephuser
```

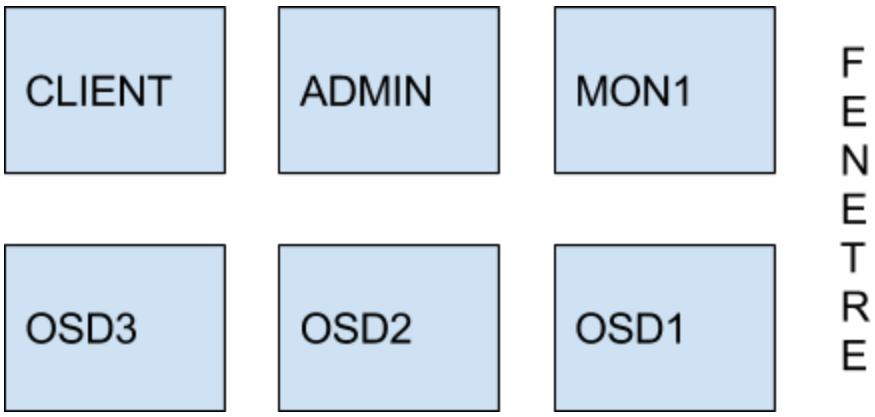
```
Host osd2
  Hostname grapcli10
  User cephuser
Host osd3
  Hostname grapcli08
  User cephuser
Host client
  Hostname grapcli07
  User cephuser
```

Une fois ces étapes préliminaires de configuration faites, nous avons utilisé l'utilitaire ceph-deploy pour installer les démons ceph sur les hôtes de notre cluster. C'est pour pouvoir utiliser ceph-deploy que nous avons crée des user sur nos machines possédant les droits sudo sans mot de passe. En effet, la machine admin se connecte automatiquement aux machines listées comme faisant partie du cluster en ssh, et fais les manipulations nécessaires à l'installation.

Il faut d'abord créer un moniteur initial et ensuite utiliser ceph-deploy tout simplement.

ceph-deploy mon create-initial
ceph-deploy disk list osd1 osd2 osd3

Voici un diagramme qui représente le cluster obtenu :



Une fois cela terminé, nous avons utilisé le mode block (rados RBD) pour créer un VBD et stocker un fichier sur le cluster et tester les fonctionnalités de celui-ci.

Conclusion

Tout au long du semestre nous avons appris les concepts des systèmes distribués. CEPH nous a permis d'en voir une application concrète. Nous avons pu alors nous familiariser avec cette technique de stockage de données. Nous avons vu ses avantages notamment en terme de récupération de données lors de pannes. Nous avons étudié les algorithmes sur lesquels s'appuie CEPH afin de garantir une répartition des données viable et cohérente. Nous avons cependant trouvé compliqué la configuration de CEPH. Cela nous a pris beaucoup de temps au niveau de l'installation même, En plus de la compréhension du fonctionnement du système de stockage de CEPH en lui-même, nous avons également beaucoup appris sur l'installation et la configuration d'un système sur des machines Linux. Il nous apparaît évident qu'un système de stockage est un outil primordial pour les entreprises.