Université de Technologie de Compiègne

Génie informatique

Rapport d'étude de SR05

Étude du système de stockage distribué CEPH

Professeur : Bertrand Ducourthial

Groupe : Duchemin, Rondelet, Coste, Mombrun, Dupont, Dilly, Ollive, Chamaux

Printemps 2017

Last update : 28 juin 2017

RÉSUMÉ

Les technologies de l'information et de la communication subissent, depuis plusieurs années, une profonde transformation. Il est désormais commun de croiser des termes génériques tels que *cloud*, *big data* ou *IaaS* [1].

Ces termes revêtent le même besoin sous-jacent : la nécessité pour les systèmes concernés d'être **disponibles**, **performants**, **accessibles** et **évolutifs**. En particulier, ces systèmes s'appuient tous sur une composante de **stockage** qui conditionne leur bon fonctionnement.

Tandis qu'auparavant, les données étaient souvent centralisées sur un seul outil de stockage – qu'il s'agit d'un serveur unique ou d'une ferme de serveurs –, cette solution n'est plus viable pour ces nouvelles technologies.

En effet, d'une part, le fait qu'une donnée ne soit accessible que par un seul biais induit l'existence systématique de *goulots d'étranglement*. D'autre part, le besoin de disponibilité ne tolère pas les pannes ni la perte de données.

Pour palier les inconvénients des systèmes de stockages classiques, de nombreuses solutions dites de **stockage distribué** sont apparues. Dans ce document, nous justifions de la pertinence de ces solutions et nous concentrons sur une technologie en particulier : **Ceph**.

Après avoir étudié ses fondements conceptuels et les algorithmes déployés, nous démontrons son bon fonctionnement en simulant une panne en situation réelle.

1. Pour Infrastucture as a Service, un dérivé du *cloud computing* permettant de louer une infrastructure distante.

TABLE DES MATIÈRES

1 INTRODUCTION

Il existe de très nombreuses solutions de stockage distribué. Sous ce terme se dessinent plusieurs concepts qu'il s'agit d'éclaircir avant d'étudier une technologie en particulier.

Dans cette courte partie, nous présentons les caractéristiques générales que partagent les systèmes de stockage distribués et justifions de leur pertinence au regard des technologies actuelles.

Nous nous concentrons ensuite sur Ceph, un système de stockage distribué libre, fiable, évolutif et performant. Nous expliquons les concepts spécifiques qu'il embarque et les nouvelles possibilités qu'il offre.

Enfin, nous le comparons aux principales solutions existantes et expliquons sa valeur ajoutée sur le marché.

1.1 LES ENJEUX DU STOCKAGE DISTRIBUÉ

Comme nous l'avons vu, le stockage distribué répond essentiellement à la direction que prennent les technologies modernes. Dans cette section, nous exposons les différents enjeux auxquels répond le stockage distribué.

1.1.1 Haute disponibilité et rapidité

La **haute disponibilité** se définit comme la période temporelle durant laquelle un service est disponible et *utilisable*, et par extension le temps nécessaire au système pour répondre à la requête d'un utilisateur. Un système est dit **hautement disponible** s'il est suffisamment robuste face aux pannes, lui permettant de fonctionner en cas de problèmes internes.

Le stockage distribué adresse – plus ou moins selon les cas – cette problématique. En particulier, la présence de plusieurs machines permet de tolérer certaines pannes,

d'autres prenant alors le relais. De plus, l'utilisation de la redondance ou de la réplication [1] tolère les défaillances mémoire sans nécessiter d'intervention humaine, comme on peut le voir avec les mécanismes de *backup* des systèmes centralisés.

Pareillement, les solutions de stockage centralisées, même à plusieurs, échouent à égaler le temps nécessaire pour satisfaire une requête des systèmes distribués. La centralisation induit systématiquement un goulot d'étranglement à un certain point.

1.1.2 Mise à l'échelle et flexibilité

Les systèmes distribués permettent intrinsèquement une mise à l'échelle (en anglais, *scalability*). En effet, les systèmes étant conçus pour résister à des pannes, comme l'extinction transitoire d'une des composantes, il est par là même conçu pour tolérer l'ajout d'autres de ces composantes.

La modification de l'infrastructure du système, que ce soit au niveau du nombre de composantes ou de leurs configurations, est alors beaucoup plus aisée qu'avec une architecture centralisée. Aussi, ces composantes peuvent avoir plusieurs rôles en plus du stockage, leur permettant d'être utilisées dans d'autres applications si besoin.

Ce besoin est fondamental, car les architectures modernes sont amenées à évoluer très rapidement pour satisfaire la demande. L'industrie se tourne en particulier de plus en plus vers des outils permettant une mise à l'échelle « automatique ».

1.1.3 Coûts

Aujourd'hui, des quantités phénoménales de données sont générées a chaque instant, et nécessitent d'être stockées – au moins l'espace de quelques instants – pour être analysées. Cette évolution fulgurante de la quantité de données à traiter va continuer avec l'ère des **objets connectés**.

Dans le cadre d'un système centralisé, les prix augmentent considérablement dès lors que la capacité de stockage est importante. Au contraire, les systèmes distribués utilisent généralement des machines « classiques », *i.e.* contenant un processeur, des disques et une interface réseau. Ces machines sont parfois capables de participer à l'intelligence collective du système.

1. Il s'agit du fait de disposer de plusieurs copies de la même donnée. Les solutions de stockage distribué mettent quasiment toutes en place cette solution, en conservant des copies de la donnée (*réplicats*) sur plusieurs machines physiques.

Au total, le coût de l'ensemble de ces machines est bien inférieur au coût d'une « super-machine » de stockage et de contrôle pour des performances égales.

Enfin, l'investissement dans un système distribué n'est pas perdu si le besoin de stockage diminue, car les composantes du système sont ré-utilisables pour toute autre application, à l'inverse des solutions hautement spécialisées et centralisées.

1.2 INTRODUCTION À CEPH

Ceph [2] est la solution de stockage distribuée qui fait l'objet de la suite de cette étude. Elle tire son origine des travaux de S. Weil pour sa thèse ; ce dernier continuera à l'améliorer à plein temps par la suite.

Ceph vise à s'imposer comme la solution moderne pour le stockage distribué et se définit comme **fiable**, **performant** et **évolutif**. Notons que Ceph est totalement **open-source**. Nous ne présentons pas dans cette section ses concepts sous-jacents, mais seulement quelques grandes caractéristiques.

1.2.1 Caractéristiques

Élimination des points uniques de défaillance

La suppression des points uniques de défaillance (*single point of failure* ou SPOF en anglais) est l'une des pierres angulaires de Ceph. Un point unique de défaillance est un élément d'un système dont le reste du système dépend. En d'autres termes, une panne de cet élément entraîne l'arrêt du système.

C'est le cas des systèmes centralisés, mais aussi de certaines solutions de stockage distribuées. Ceph vise donc la très haute disponibilité ainsi qu'une tolérance à de nombreuses pannes sur des composants quelconques.

En particulier, la suppression complète des composants uniques permettant de calculer l'emplacement des données stockées confère à Ceph une grande fiabilité.

Utilisation de l'intelligence collective

Une autre des forces de Ceph est l'idée que chaque composante du stockage est une unité *potentiellement intelligente*. En d'autres termes, le fait de posséder un processeur,

2. Ce nom tire son origine du surnom donné à certains céphalopodes, et tire son origine du grec ancien. Ce nom suggère en particulier la construction extrêmement parallèle de Ceph.

une interface réseau et de la mémoire permet d'exécuter d'autres actions que la simple réponse à des ordres de lecture et d'écriture.

Ceph s'attache alors à déporter au maximum les algorithmes utilisés au sein même des composantes de stockage, afin que tous puissent participer à l'effort collectif, et par là même, réduire les coûts et améliorer la performance.

Différents niveaux de stockage

Ceph propose d'utiliser une même grappe de composantes de stockage pour gérer trois niveaux de manière **unifiée** :

- Le niveau **bloc**, où le système de stockage est vu comme un unique **périphérique bloc**, à la manière d'un disque dur local. Il est alors possible de lire et d'écrire de façon standard sur le périphérique et la réplication est assurée automatiquement.
- Le niveau **objet**, plus traditionnel, où une analogie existe avec l'idée de base de données NoSQL. Chaque donnée est manipulée comme un objet et permet d'abstraire le système de stockage *effectivement* utilisé. Notons que Ceph est compatible avec des solutions propriétaires comme Amazon S3 ou Swift.
- Le niveau **système de fichiers**, proposant une interface compatible POSIX.

Ces trois niveaux étant unifiés au sein du même système, il est possible de mutualiser les ressources de Ceph pour des utilisations très complémentaires.

1.2.2 Utilisateurs

Il n'existe *plus* de liste publique d'entreprises ou d'organismes utilisant Ceph. Cependant, Ceph est près pour la mise en production à tous les niveaux, plus particulièrement depuis que CephFS – permettant de mettre en place le niveau système de fichiers – est stable.

On peut tout de même noter qu'OVH utilise Ceph dans la quasi-totalité de ses offres de stockage. D'autres acteurs majeurs tels que Yahoo, CloudWatt et Orange utilisent Ceph en production. Des entreprises telles que Deutsche Telekom, le Crédit Mutuel, Thales et Ubisoft s'intéressent de près à Ceph.

Cet intérêt soutenu de l'industrie renforce la crédibilité de Ceph en tant qu'outil moderne, adapté et performant.

Notons enfin que le CERN a validé Ceph comme futur de son architecture informatique, un système virtualisé et redondant. En particulier, les données incroyablement massives du LHC, par exemple, génèrent 30 pétaoctets de données par an. Ceph peut gérer jusqu'à l'exaoctet, et les tests du CERN ont démontré sa robustesse et sa fiabilité.

1.3 COMPARAISON AVEC LES AUTRES SYSTÈMES DE STOCKAGE DISTRIBUÉ

Dans cette section, nous comparons brièvement Ceph avec les autres solutions de stockage distribué et expliquons les nouveautés apportées.

Nous ne traitons pas des systèmes de fichiers antiques ou peu utilisés, dont les inconvénients sont évidents [3] mais tentons de le comparer aux technologies plus modernes.

StorNext et EMC ScaleIO sont par exemple des solutions de stockage distribué propriétaires qui proposent une mise à l'échelle performantes, mais n'implémente pas de stockage unifié de niveau bloc, objet et système de fichiers.

GPFS permet d'atteindre d'excellentes performances, comprend la tolérance aux pannes et la redondance, mais n'est pas open-source. De plus, les systèmes similaires – dits SAN pour Storage Area Network – utilisent des *disques durs en réseau*, et ne tirent absolument pas partie de l'intelligence collective. Ils comprennent en outre au moins un SPOF en le gestionnaire central qui coordonne l'accès aux disques.

D'autres solutions comme Lustre ou OpenZSF ont pris le parti de stocker les données en tant qu'objets plutôt qu'en tant que blocs. Ces objets intègrent en particulier les métadonnées propres aux données concernées. L'intelligence collective est alors possible. Pour autant, des systèmes comme Lustre ne délèguent que très peu de tâches aux composantes de stockage.

Gluster est une solution moderne très utilisée, mais ne propose pas de stockage unifié. Les performances de Ceph sont en général supérieures à celles de Gluster.

Swift, un des concurrents les plus sérieux de Ceph, ne propose pas non plus de stockage unifié. Ceph est plus performant, mais notons que le fonctionnement de Swift

3. Mauvaise gestion de la réplication, existence systématique de SPOF...

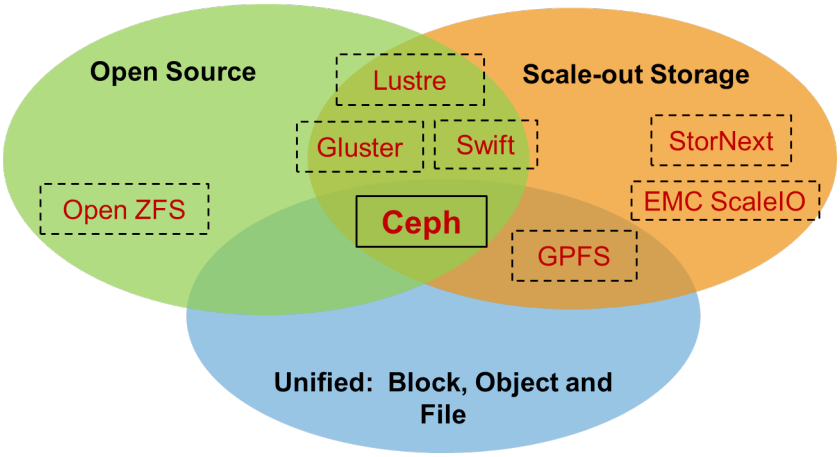

FIGURE 1 – Comparaison de Ceph et des autres solutions de stockage distribué.

propose une sécurité supplémentaire. En effet, Ceph est sensible à la compromission d'une composante de stockage qui peut alors directement écouter tout le trafic non-chiffré. Swift, quant à lui, permet de séparer le réseau des « clients » et celui des « serveurs ».

La figure 1 résume la position de Ceph par rapport aux autres solutions. Elles ne sont pas toutes citées ici.

Ceph pourrait bien être une solution unique en offrant un ensemble de caractéristiques qu'aucune autre solution ne combine. Open-source, aisément évolutif, performant et extrêmement robuste, la sortie de la confidentialité du milieu universitaire pour être mis en production dans l'industrie témoigne de l'intérêt croissant pour Ceph.

2 ARCHITECTURE LOGIQUE ET PHYSIQUE DE CEPH

Après avoir présenté Ceph, ses caractéristiques et ses différences avec les méthodes de stockage distribué existantes, cette partie vise à décrire le fonctionnement et l'architecture d'un cluster Ceph.

2.1 ARCHITECTURE GÉNÉRALE

Afin de pouvoir tirer profit de l'infrastructure de stockage distribué offerte par Ceph, de nombreuses interfaces ont étés développées (cf figure 2) afin d'offrir un large panel de possibilités aux utilisateurs. Ces interfaces sont des points d'accès au même composant central assurant la gestion des données distribuées : **RADOS** [1]. Cet élément est crucial et gère toute la logique liée à la distribution et la réplication des données. De ce fait, une partie lui sera dédiée plus tard dans le présent rapport.

Nous présentons d'abord les différents éléments composants Ceph, puis détaillons en profondeur les concepts.

2.1.1 Librados

Afin de pouvoir construire des applications basée sur RADOS, la bibliothèque **Librados** a été développée dans le but de pouvoir interagir au plus bas niveau avec le cluster Ceph. Cette bibliothèque écrite en langage C permet en particulier d'accéder directement et de façon concurrente au cluster. Des bilbliothèques similaires ont étés développées dans d'autres langages comme Python, Java et C++.

L'utilisation de librados pour communiquer directement avec RADOS améliore drastiquement les performances des applications tout en leur offrant un maximum de fiabilité et de flexibilité.

1. RADOS signifie Reliable Autonomic Distributed Object Store, littéralement Stockage d'objets fiable, automatique et distribué.

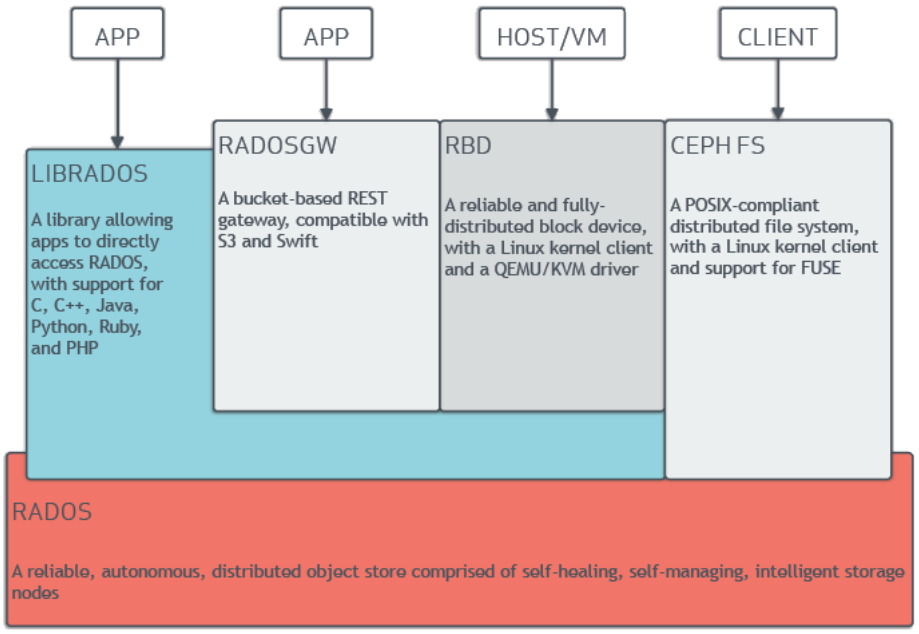

FIGURE 2 – Aperçu de l'architecture générale de Ceph

2.1.2 Interfaces offertes par Ceph

Outre l'interface native de Ceph, il existe d'autres modes d'accès à RADOS. Avant de détailler ces interfaces, il est important de signaler qu'il est possible d'utiliser **plusieurs interfaces** pour un même cluster, il n'est par exemple pas nécessaire de configurer trois clusters différents pour trois interface.

RADOS BLOCK DEVICE (RBD) Ce dispositif découpe une image de **périphérique bloc** sur plusieurs objets dans le cluster de stockage Ceph, où chaque objet est mappé sur un groupe de placement [2] et distribué.

Ces techniques se révèlent être des options intéressantes pour la virtualisation et le cloud computing. Dans les scénarii basés sur des machines virtuelles, il est de mise de déployer en général ce dispositif avec le pilote de stockage réseau `rbd` dans QEMU [3]

2. Nous y reviendrons, mais disons simplement qu'il s'agit d'une entité permettant d'agréger des objets.
3. QEMU est un logiciel libre de machine virtuelle.

/ KVM[4], où l'ordinateur hôte utilise la librairie librbd[5] pour fournir un service de périphérique bloc au client.

RADOSGW RADOSGW (RADOS Getaway) est une interface de **stockage d'objets** construite au dessus de librados pour fournir aux applications une passerelle RESTful[6] aux clusters de stockage Ceph. Deux interfaces sont possibles :

- S3 - compatible : fournit une fonctionnalité de stockage d'objet avec une interface compatible avec un grand sous-ensemble de l'API Amazon S3[7] RESTful.
- Swift-compatible : fournit une fonctionnalité de stockage d'objet avec une interface compatible avec un grand sous-ensemble de l'API OpenStack Swift[8].

L'utilisation couplée de ces deux API est possible. Par exemple, vous pouvez écrire des données à l'aide de l'API compatible S3 avec une application, puis lire des données à l'aide de l'API Swift compatible avec une autre application.

CEPHFS Le **système de fichiers** Ceph (CephFS) est un système de fichiers respectant la norme POSIX qui utilise un cluster de stockage Ceph pour stocker les fichiers. Les fichiers CephFS sont mappés sur des objets que Ceph stocke dans le cluster de stockage. Les clients Ceph ont le choix de monter un système de fichiers CephFS comme un objet kernel ou un système de fichiers dans un espace utilisateur (FUSE).

Ce service inclut le serveur de métadonnées Ceph (MDS) déployé avec le cluster Ceph Storage. L'objectif du MDS est de stocker toutes les métadonnées du système de fichiers (répertoires, propriété des fichiers, modes d'accès, etc.) dans les serveurs de Ceph haute disponibilité où les métadonnées se trouvent dans la mémoire. L'idée derrière le MDS est que les opérations simples du système de fichiers comme l'affichage du contenu d'un répertoire ou la modification d'un répertoire (ls, cd) ne doivent pas surcharger inutilement les démons Ceph OSD. Ainsi, la séparation des métadonnées

4. KVM est un hyperviseur de type I sous Linux.
5. Fournit un accès en forme de fichier aux images RBD.
6. API Restful : interface de programmation d'application qui fait appel à des requêtes HTTP pour obtenir (GET), placer (PUT), publier (POST) et supprimer (DELETE) des données.
7. Amazon S3 : site d'hébergement de fichiers offert par Amazon Web Services.
8. Système de stockage de données d'OpenStack.

des données signifie que le système de fichiers Ceph peut fournir des services performants sans surcharger le cluster de stockage Ceph.

2.2 RADOS COMME SOCLE DE CEPH

Après avoir défini les enjeux du stockage distribué et avoir examiné rapidement l'architecture de Ceph, nous présentons dans cette section les caractéristiques majeures de RADOS, le composant central d'un cluster Ceph.

Afin de bien comprendre la philosophie sous-jacente de RADOS, il est fondamental de bien saisir les **motivations** de la création de ce composant. Comme nous l'avons vu dans l'introduction, Ceph est né du constat que les systèmes de stockage distribués « classiques » sont peu adaptés au stockage de volumes de données très importants, de l'ordre du petaoctet (Po) ou de l'exaoctet (Eo) [9].

En effet, dans le modèle traditionnel de stockage distribué, le fonctionnement entier du cluster dépend de **métadonnées** [10]. Ces dernières sont fondamentales car elles permettent de localiser les données stockées sur le cluster. Leur perte ou leur altération résulte fréquemment en une impossibilité partielle ou totale d'utiliser le cluster – une telle situation est rédhibitoire pour une entreprise. Ainsi, les métadonnées agissent comme un **SPOF** [11], soit comme un risque de faille considérable pour la totalité de l'infrastructure de stockage.

Le statut sensible de ces méta-données nécessite de leur apporter un soin particulier sur le cluster, quitte les répliquer plusieurs fois. Il s'agit d'une véritable problématique, car si stocker une quantité très importante de données s'avère être un défi complexe en soi, stocker et protéger les méta-données contre les pertes éventuelles constitue un challenge supplémentaire duquel on aimerait s'affranchir. Il apparaît alors évident, à ce stade, que le modèle « traditionnel » utilisant et stockant des métadonnées pour assurer le stockage distribué d'objets présente deux problèmes majeurs :

- D'une part, la présence d'un SPOF est une véritable erreur de conception par rapport aux pré-requis d'un système de stockage distribué ;

9. Respectivement 10^{15} et 10^{18} octets.
10. Donnée dont l'information porte sur une autre donnée.
11. Pour Single Point Of Failure, un point d'un système informatique dont le reste du système est dépendant et dont une panne entraîne l'arrêt complet du système.

- D'autre part, la solution consistant à répliquer toutes les méta-données est lourde et représente un véritable frein aux bonnes performances du cluster.

C'est suite à de tels constats qu'est né RADOS.

L'objectif de RADOS est, entre autres, d'assurer le bon fonctionnement d'un cluster de stockage tout en s'affranchissant des métadonnées « parasites » qui rendent la montée en charge et la mise à l'échelle du cluster très difficiles.

RADOS se base sur l'idée qu'un système de stockage réparti peut être soumis à un nombre important de pannes : les appareils changent d'état de disponibilité de manière continuelle. Ainsi, plutôt que de stocker des métadonnées *en permanence*, les créateurs de Ceph ont eu l'idée de pouvoir les calculer « **à la demande** ». Pour ce faire, RADOS repose sur un algorithme [12] qui assure la répartition pseudo-aléatoire [13] des données à travers le cluster.

L'unité de stockage utilisée par RADOS est l'**objet**. Cette approche consiste à manipuler des unités de données discrètes étiquetées par un identifiant unique. Ces objets sont gardés au sein d'un dépôt unique et stockés dans un espace d'adressage plat [14], plus communément appelé « **storage pool** ». Les objets ne sont pas – comme peuvent l'être les fichiers – imbriqués dans une succession de dossiers. N'étant plus dans le cadre d'un système hiérarchique, les développeurs de RADOS ont dû introduire différentes **unités logiques** leur permettant de gérer la répartition d'objets distribués de façon optimale.

La suite de cette partie vise à décrire ces structures de données, dans le but d'expliquer ensuite le fonctionnement interne de RADOS.

12. Cet algorithme, appelé CRUSH, fait l'objet de la section 3 dans laquelle il sera traité en profondeur.
13. L'aspect pseudo-aléatoire de CRUSH signifie qu'à première vue, l'assignation d'un espace de stockage pour un objet semble être aléatoire, mais en réalité ce calcul est tout à fait déterministe et renvoie les même résultats lorsqu'il est calculé avec les même paramètres d'entrée.
14. Un espace de stockage plat (ou *flat address space*) est un espace mémoire où chaque adresse (incrémentée de une unité en une unité, allant de 0 jusqu'à la fin de l'espace mémoire) représente une unité de donnée, *e.g.* un objet.

2.2.1 Les objets

Les objets sont les plus petites unités de données dans Ceph. Ils contiennent des données à l'état binaire ainsi que les métadonnées associées, et sont étiquetées par un identifiant unique sur l'ensemble du cluster. Les objets Ceph ne sont pas restreints par une taille maximale ; ils sont stockés et répliqués sur l'ensemble du cluster. La figure 3 illustre la structure d'un objet.

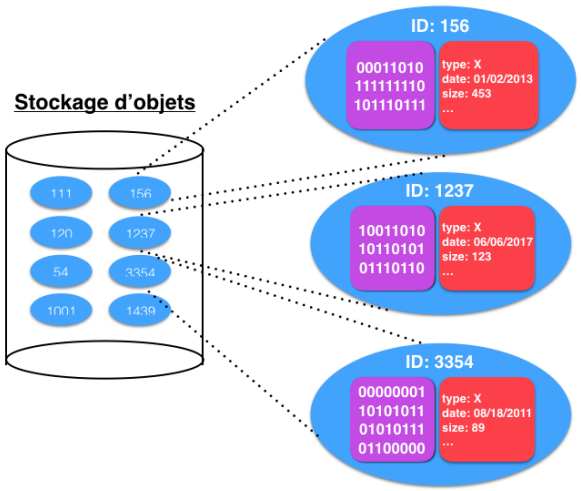

FIGURE 3 – Aperçu du stockage d'objets.

2.2.2 Les groupes de placements

Plutôt que de manipuler chaque objet de façon indépendante, ce qui serait lourd et peu efficace, des unités logiques nommées « **groupes de placements** » (en anglais, *placement group* ou PG) agrègent plusieurs objets. En effet, bien que chaque objet soit identifiable par un identifiant unique, dans le cas de clusters stockant des volumes de données très importants, manipuler des millions de données indépendantes serait une tâche très coûteuse. En particulier, identifier un objet spécifique, le répliquer, et assurer une cohérence entre toutes les versions serait rédhibitoire en termes de performances. Pire encore, chaque machine ajoutée au cluster et/ou chaque nouvelle donnée stockée complexifierait davantage la manipulation d'objets.

En ce sens, il est naturel de grouper un sous-ensemble d'objets dans une structure logique ; cela permet de fortement diminuer la complexité calculatoire nécessaire à la

manipulation des données. Il n'est plus question de stocker et de dupliquer un objet spécifique à chaque fois, mais plutôt de manipuler des sous ensembles d'objets [15].

2.2.3 Les files

Une file (ou *pool* en anglais) est une partition logique [16] dans laquelle un utilisateur de RADOS peut stocker des objets. Chaque pool contient de nombreux *placement groups*, et est répliquée sur *certains* nœuds du cluster. La pérennisation des données est assurée via l'utilisation de techniques de redondance telles que la réplication ou le code d'effacement (*erasure coding* ou EC) [17].

2.2.4 Appareils de stockage d'objets

Les appareils de stockage d'objets (*Object Storage Devices* ou OSD) constituent l'un des grands concepts introduits par Ceph. Jusqu'alors, les nœuds des clusters de stockage étaient simplement destinés à stocker des données. De ce fait, toute la logique était présente dans une poignée de **nœuds maîtres** (ou *master*) qui s'occupaient de gérer toute la logistique interne au cluster, et qui ordonnaient aux **nœuds esclaves** (ou *slave*) d'opérer sur les données.

Dans RADOS, en revanche, les OSD sont des appareils de stockage tirant profit de leur **intelligence** inhérente ; ils sont une composante à part entière de la logique du système reparti. Leur structure est la suivante :

- Un processeur (CPU) ;

15. Dans la suite de ce rapport, nous utiliserons fréquemment des abus de langage en parlant de stockage ou de réplication de *placement groups*. En réalité, ce sont les objets inclus dans les groupes de placement qui seront stockés et/ou répliqués. En effet, ce sont les objets qui contiennent les données du cluster. Les *placement groups* sont simplement présents dans le but de réduire la complexité de l'infrastructure.

16. En d'autres termes, les pools ne concernent pas des zones mémoires contiguës, mais une vision de plus haut niveau.

17. La technique d'Erasure Coding introduite dans l'une des dernières version de Ceph permet de répondre aux défauts de la technique de réplication utilisée depuis des années. En effet, dans le cas de volumes très importants de données, répliquer ces dernières augmente significativement le besoin en stockage d'un cluster. Cette technique présente donc ses limites. L'idée, ici, est de diviser les objets, d'en encoder chaque fragment, d'y ajouter des informations utiles à la restauration des données, et de stocker chacun de ces morceaux sur des machines différentes. De cette manière, en cas de mise hors service d'un/plusieurs nœud(s) du cluster, un sous ensemble des morceaux d'un objet subsistera et permettra de restaurer la totalité de l'objet. Le coût de l'*erasure coding* représente 40% du coût de la réplication. C'est une amélioration substantielle qui allège le coût de la redondance des données tout en gardant un résultat similaire à celui de la réplication

- De la mémoire vive (RAM) ;
- Une interface réseau ;
- Un espace de stockage (disque dur ou RAID [18]).

Pour assurer le fonctionnement du cluster, ces derniers communiquent à l'aide de protocoles de type pair-à-pair [19].

Dans le but de conserver des performances optimales, RADOS se base sur un ensemble de structures logiques facilitant la manipulation de grandes quantités d'objets distribués à travers le cluster. Ces structures interagissent entre elles pour garantir la cohérence des données dupliquées et la reprise après erreur lorsqu'un problème survient sur le cluster.

L'étude des mécanismes internes à RADOS est abordée dans la suite de cette partie.

2.2.5 Gestion de la répartition des objets

Afin de gérer la répartition des données dans le cluster, RADOS utilise un **algorithme de placement** nommé **CRUSH** [20]. Cet algorithme permet, à partir d'un identifiant d'objet ou d'un groupe de placement, de **calculer** la liste des OSD responsables de leur stockage. En ce sens, l'algorithme CRUSH permet de déterminer des métadonnées « à la demande », permettant d'assigner [21] des groupes d'objets à des OSD tout en repartissant la charge de stockage de façon uniforme sur l'ensemble du cluster.

18. La technologie RAID (Redundant Array of Independent Disks), ou « Ensemble redondant de disques indépendants », consiste à mettre en grappe plusieurs disques durs pour en obtenir une partition logique unique. Ainsi, à partir de plusieurs disques durs physiques, on obtient un seul espace visible par le système d'exploitation. En fonction de l'objectif, le RAID permet d'accroître la performance d'accès et d'écriture des données ou d'améliorer la sécurité des informations.

19. La particularité des architectures pair-à-pair réside dans le fait que les données puissent être transférées directement entre deux postes connectés au réseau, sans transiter par un serveur central. (source : https://fr.wikipedia.org/wiki/Pair_à_pair)

20. CRUSH signifie Controlled Replication Under Scalable Hashing, littéralement « Réplication contrôlée par hachage extensible ». Les objectifs de cet algorithme sont brièvement présentés dans cette partie. La partie 3 est dédiée à l'étude approfondie de ses caractéristiques.

21. L'assignation de groupes de placements aux OSD se fait *pseudo-aléatoirement*. En d'autres termes, l'observation extérieure des résultats de l'algorithme semble indiquer que la répartition des données est totalement aléatoire dans le cluster. En revanche, CRUSH est un algorithme déterministe, et s'assure de répartir la charge de travail équitablement sur l'ensemble des nœuds du cluster.

Gestion interne du cluster et cohérence des données

Alors que CRUSH permet de calculer la position des données dans l'infrastructure de stockage, ce sont les OSD, qui, ensuite, assurent le **fonctionnement** du cluster. En effet, dans un cluster de milliers de machines, les pannes et les reprises après erreurs peuvent être très fréquentes. De ce fait, l'état du cluster est amené à changer très régulièrement. Pour en faciliter la gestion, cet état est entièrement représenté par une **carte**, la *cluster map* [22], faisant l'inventaire de tous les OSD étant dans le cluster.

Plutôt que de n'être présente que sur un nœud spécifique – ce qui introduirait un point de faille unique dans l'infrastructure –, cette structure est dupliquée sur tous les nœuds du cluster, ainsi que sur les clients qui interagissent avec RADOS. Il serait impensable, en revanche, de devoir mettre à jour la carte du cluster sur l'ensemble des nœuds chaque fois qu'un OSD devient indisponible ou est ajouté. L'opération serait bien trop coûteuse et diminuerait significativement les performances globales, agissant comme un *goulot d'étranglement*.

Pour adresser cette problématique, la carte est **datée** par une époque [23] représentant sa version [24]. Ajouter l'époque de la carte détenue par chaque OSD aux messages qu'il émet permet aux autres nœuds du cluster de se mettre d'accord sur la distribution des données à l'instant de leur communication. En d'autres termes, l'échange de leur version de la carte leur permet de savoir si leur information est à jour, ou si elle nécessite d'être enrichie. De cette manière, les changements d'état du cluster peuvent être transmis de proche en proche à mesure que les OSD communiquent. Cette méthode de mise à jour des nœuds du cluster, appelée **lazy update** [25], permet de ne pas surcharger le réseau interne à l'infrastructure. Les mises à jour sont propagées de proche en proche en cas de besoin.

En revanche, pour garder un état global cohérent, ces actualisations doivent tout de même circuler relativement rapidement d'un OSD à un autre. Pour ce faire, les OSD émettent des messages dits « **heartbeat** », ayant la double particularité pour un

22. En réalité, cette *cluster map* est un conglomérat de « cartes » différentes. Elle englobe la carte des OSD, la carte des groupes de placement, la carte de CRUSH, la carte des moniteurs, et, enfin, la carte des serveurs de métadonnées.

23. Notons que l'utilisation d'une époque (*epoch* en anglais) pour gérer les versions de la carte du cluster fait écho aux horloges de Lamport (estampilles) que nous avons étudié en SR05.

24. Plus la valeur de l'époque est petite, plus la carte du cluster est ancienne.

25. Littéralement « mise à jour fainéante », faisant référence à l'absence de mise à jour systématique, comme dans l'algorithme de *snapshot* avec lestage.

nœud de savoir si ses voisins sont toujours en train de s'exécuter [26], et de recevoir la version de la carte du cluster de ses voisins de façon fréquente et régulière. Avec cette technique, les OSD peuvent fréquemment demander les mises à jour incrémentales [27] nécessaires à leur remise à jour.

Enfin, en cas de non-réception répétée des messages de *heartbeat* d'un voisin, un OSD peut prendre la décision de notifier le problème à un nœud dit **moniteur**. Ces nœuds n'ayant jusqu'alors pas été mentionnés ont pour rôle général de **conserver la version maître** de la carte du cluster. Pour ce faire, ils **votent** un consensus permettant de conserver une version **stable** et la plus **à jour** possible de la carte maître. En particulier, l'état d'un nœud soupçonné « hors service » sera mis à jour par leurs soins, et l'OSD sera répertorié comme étant *down* [28], avant que les placement groups qu'il stockait soient confiés à d'autres OSD du cluster.

« CRUSH lookup » et interactions avec les utilisateurs

Lorsqu'un utilisateur souhaite interagir avec RADOS, ce dernier doit se munir d'une carte du cluster pour avoir une idée de la répartition des données et avoir connaissance des OSD qui constituent l'infrastructure. Pour ce faire, l'utilisateur initie l'échange avec les nœuds **moniteurs** [29] pour obtenir une carte du cluster. Il pourra alors stocker des objets dans une *pool* et utiliser l'algorithme CRUSH pour déterminer l'emplacement de stockage de la donnée. Cette étape se nomme **CRUSH lookup**, et nécessite la détention de la carte du cluster.

26. *i.e.* l'OSD voisin n'est pas devenu indisponible suite à une erreur.

27. Lorsque deux cartes de cluster ($c1$ et $c2$ avec $c1.\mathrm{epoque} > c2.\mathrm{epoque}$) n'ont pas la même version, cela signifie que l'une est plus récente que l'autre. De cette manière, le nœud ayant l'époque la plus faible peut demander à son voisin de le mettre à jour, et de l'informer des changements qu'a subit l'infrastructure durant la période de temps $c1.\mathrm{epoque} - c2.\mathrm{epoque}$. Cette mise à jour est faite en envoyant une succession de messages dont chacun décrit un événement ayant eu lieu sur l'infrastructure, et ce jusqu'à ce que l'OSD le moins à jour ait une époque égale à $c1$. De cette manière, chaque OSD est assuré d'avoir en sa connaissance chaque événement étant arrivé sur le cluster. L'historique est connu de tous les nœuds du cluster et chaque distribution intermédiaire de donnée est prise en compte. Cette technique élégante donne les moyens à chaque OSD de mettre à jour ses voisins, et la cohérence des données est assurée.

28. À savoir que l'état d'un OSD est défini par un couple dont le premier membre indique si l'appareil est joignable (*up*) ou pas (*down*), et dont le second membre renseigne sur l'inclusion du nœud dans la carte du cluster (*in* ou *out*).

29. Pour les interactions suivantes, l'utilisateur pourra directement communiquer avec les OSD. Sa copie locale de la *cluster map* recevra les modifications incrémentales nécessaires à sa remise à jour si son époque est trop faible. Cela évite au cluster de devoir mettre les clients à jour à mesure que l'état interne évolue.

L'emplacement déterminé par le CRUSH lookup est égal à l'identifiant de l'OSD sur lequel sera faite la demande en écriture de l'objet à stocker. Cet OSD est dit **primaire** (ou *Primary OSD*)[30] pour le placement group. Son rôle est de prendre en charge la gestion de la répartition du placement group au sein du cluster.

Le processus de stockage d'un objet par un utilisateur peut être résumé par la séquence suivante :

1. L'objet est placé dans un placement group ;
2. Le placement group est placé dans une file ;
3. Le mécanisme de CRUSH lookup calcule l'identifiant de l'OSD primaire sur lequel écrire le PG ;
4. L'OSD en question, une fois la demande reçue, utilise la version de la carte du cluster qu'il détient pour répliquer le placement group n fois[31].
5. L'OSD primaire renvoie un message d'**acquittement** (« ACK ») à l'utilisateur après avoir reçu **tous** les acquittements des OSD *secondaires*[32]. Cet ACK est caractéristique d'une réplication réussie[33].

De cette manière, le coût lié à la réplication des données est déporté au sein du réseau interne à l'infrastructure lors de sa prise en charge par les OSD du cluster. Cet ensemble d'OSD (primaire et secondaires) forme ce qui est plus communément appelé un **acting set**[34]. L'OSD primaire est donc responsable, pour un placement group, du processus de **peering**, qui vise à assurer l'intégrité des informations parmi les différents OSD de l'acting set d'un placement group.

C'est l'OSD primaire – dont l'identifiant est calculé lors du CRUSH lookup effectué par l'utilisateur – qui sera en mesure d'accepter la première requête d'écriture venant du client, et qui assurera le processus de la réplication et d'acquittement à l'utilisateur.

30. Un OSD primaire est responsable, pour un placement group au sein du cluster, de s'assurer que les données du PG restent cohérentes. Notons qu'un OSD peut être primaire pour un placement group, et secondaire pour un autre ; c'est d'ailleurs ce qu'il se passe en pratique
31. Les conditions de réplications des placements group sont configurées dans la *pool*.
32. Les OSD secondaires sont ceux contenant les réplicas du placement group.
33. On notera là encore, un parallèle fort avec les notions de vagues et d'ondes vues en cours de SR05, et plus particulièrement avec l'algorithme de calcul diffus.
34. Liste ordonnée d'OSD responsables du stockage d'un placement group.

Reprise après erreurs et élections des OSD primaires

Chaque OSD sait lorsqu'il est secondaire pour un groupe de placement donné. De ce fait, en cas d'indisponibilité de l'OSD primaire, suite à une erreur quelconque, c'est l'OSD secondaire suivant immédiatement l'OSD primaire dans l'acting set qui est élu OSD primaire. Pour garantir la cohérence des données après une élection, les OSD secondaires envoient tous des messages de **notification** (ou *Notify*) à l'OSD primaire afin que ce dernier découvre son rôle d'« élu »[35], et puisse enclencher le processus de **peering**[36].

Pour assurer son rôle d'OSD primaire, le site élu contacte tous les OSD ayant fait partie de l'acting set du placement group pour lequel il est devenu responsable, dans le but de rassembler toutes les données et métadonnées associées. De cette manière, l'OSD primaire construit un historique des modifications apportées sur le placement group, et est alors en mesure de connaître la version la plus récente que chaque réplicat devrait avoir. Le site élu obtient également l'historique de tous les journaux concernant le placement group.

A l'aide de ces journaux, l'OSD primaire sait ce à quoi devrait « ressembler » le placement group sur chaque site secondaires. De ce fait, il peut, en cas de reprise après erreur d'un site de l'acting set, lui envoyer l'information nécessaire à sa reprise.

> L'OSD primaire prend en charge la gestion de la reprise après erreur concernant les données pour lesquelles il est responsable. Il peut informer chacun de ses sites secondaires de l'état que devrait avoir leur réplicas, de sorte à ce que ces derniers amorcent la démarche de reprise en arrière-plan.

De ce fait, en cas d'interruption de l'OSD primaire, c'est l'OSD secondaire qui lui succède par défaut, et, les autres OSD de l'acting set lui notifient son changement de statut. Suite a cela, le nouvel OSD primaire rassemble toutes les données lui permettant d'avoir une vue d'ensemble sur l'état de chaque répliquas du placement group.

35. L'envoi de messages de notifications des OSD secondaires vers le nouvel OSD primaire permet d'éviter à ce dernier de devoir considérer l'ensemble des placements groups qu'il stocke pour trouver celui pour lequel il est devenu primaire.

36. Le processus de peering consiste à faire en sorte que tous les OSD d'un acting set soient en accord à propos de l'état des objets inclus dans un placement group donné. Notons qu'être en accord sur l'état du placement group ne signifie en rien que tous les OSD doivent tous avoir la version la plus récente du PG.

De cette manière, le site élu est en mesure de coordonner la reprise après erreur concernant le placement group pour lequel il est responsable.

2.2.6 Les moniteurs

Le rôle principal des moniteurs Ceph consiste à conserver une copie maître de la carte du cluster. Ils fournissent également des services d'authentification et de journalisation.

Avant qu'un client Ceph puisse accéder en lecture ou écriture aux données du cluster, celui-ci doit contacter un moniteur afin d'obtenir la plus récente copie de la carte du cluster. Suite à cela, le client peut calculer l'emplacement de n'importe quel objet sur le cluster.

Un cluster de stockage Ceph peut fonctionner avec un seul moniteur. En revanche, un tel cas de figure introduit un point de faille unique. De ce fait, il est recommandé, pour tolérer les pannes, et assurer une fiabilité maximale, de configurer un ensemble impair[37] de nœuds moniteurs.

Nous détaillons le fonctionnement interne des moniteurs – en particulier dans leur mécanisme de **consensus** – dans la section 2.3

2.3 CONSENSUS DES MONITEURS

Cette section se propose d'expliquer le fonctionnement du consensus des moniteurs. En effet, dans le cas d'un groupe de moniteur, les actions de décision doivent être cohérentes. Comme l'utilisation de plusieurs moniteurs est quasi obligatoire pour garantir l'absence de SPOF, il est important d'en comprendre le fonctionnement.

2.3.1 Paxos, une solution pour surveiller l'état du cluster

Une des difficultés principales intrinsèques aux systèmes distribués est l'obtention d'une certaine fiabilité dans des décisions distribuées. Il s'agit du problème largement étudié du **consensus**. Dans notre cas, les moniteurs cherchent en permanence à se mettre d'accord sur l'**état** du cluster.

37. Quorum et split-brain.

Pour traiter ce problème, les moniteurs Ceph utilisent un algorithme de type **Paxos**, une famille de protocoles permettant de résoudre ce problème du consentement. Ces algorithmes sont connus pour être complexes ; nous en proposons une explication.

2.3.2 Comment fonctionne Paxos ?

Un nœud Paxos peut prendre l'un des trois rôles suivants, de manière non-exclusive : **proposant** *ou* leader, **accepteur** et **apprenant** :

- Un proposant (pas nécessairement unique) propose une valeur sur laquelle il souhaite un accord. Pour ce faire, il envoie une proposition contenant une valeur à l'ensemble de tous les accepteurs, qui décident d'accepter ou non la valeur.
- Chaque accepteur choisit une valeur de manière indépendante – il peut recevoir plusieurs propositions, chacune d'un autre proposant – et envoie sa décision aux apprenants, qui déterminent si une valeur a été acceptée.
- Une fois que la majorité des nœuds ont accepté, un consensus est atteint et le coordinateur diffuse un message de validation à tous les nœuds.

En pratique, un seul nœud peut prendre plusieurs ou tous ces rôles, mais dans les exemples de cette section, chaque rôle est exécuté sur un nœud distinct, comme illustré figure 4.

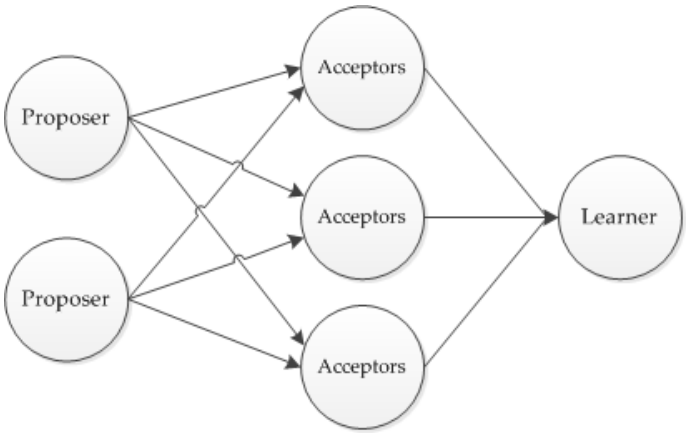

FIGURE 4 – Architecture basique de Paxos

2.3.3 Étude de Paxos à travers un exemple

Dans l'algorithme Paxos standard, les proposants envoient deux types de messages aux accepteurs : « prépare » et « accepte » les demandes. Dans la première étape de cet algorithme, un proposant envoie une demande de préparation à chaque accepteur contenant une valeur proposée (v) et un numéro de proposition (n).

Le numéro de proposition de chaque proposant doit être un entier naturel, monotone et unique au regard des numéros de propositions d'autres proposants.

Dans l'exemple illustré figure 5, il y a deux proposants, tous deux faisant des demandes de préparation. La demande du proposant A atteint les accepteurs X et Y avant la demande du proposant B, mais la demande du proposant B atteint l'accepteur Z d'abord.

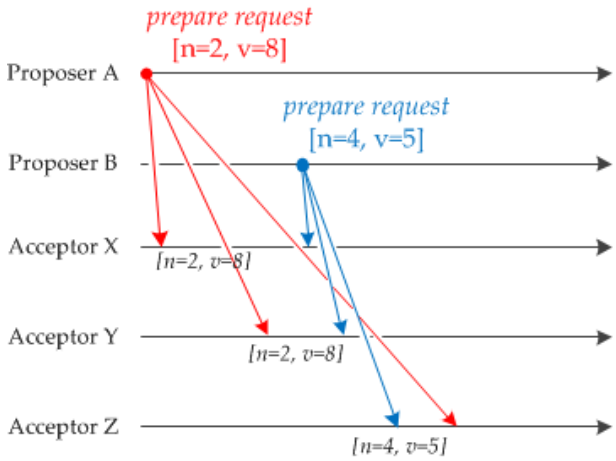

FIGURE 5 – Phase de préparation : prepare request.

Si l'accepteur recevant une demande de préparation n'a pas vu une autre proposition, l'accepteur répond avec une réponse préparatoire qui promet de ne jamais accepter une autre proposition avec un nombre de propositions inférieur. Ceci est illustré dans la figure 6, qui montre les réponses de chaque accepteur à la première demande de préparation qu'ils reçoivent.

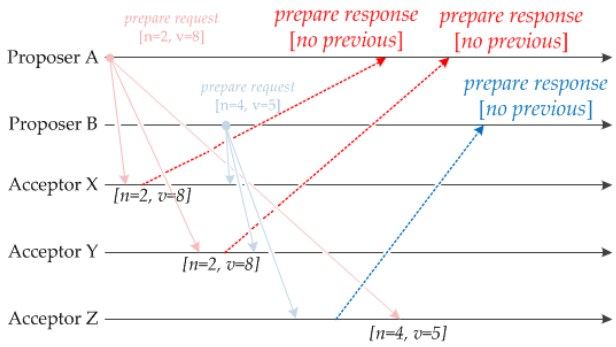

FIGURE 6 – Phase d'acceptation n°1 : prepare response.

Finalement, l'accepteur Z reçoit la demande du proposant A et les accepteurs X et Y reçoivent la demande du proposant B. Si l'accepteur a déjà vu une demande avec un numéro de proposition plus élevé, la demande de préparation est ignorée, comme c'est le cas avec la demande du proposant A à l'accepteur Z. Si l'accepteur n'a pas vu une demande numérotée plus élevée, il promet encore d'ignorer toute demande avec des numéros de proposition inférieurs, et renvoie la proposition la plus élevée qu'elle a acceptée avec la valeur de cette proposition. C'est le cas avec la demande du proposant B aux accepteurs X et Y, comme illustré figure 7.

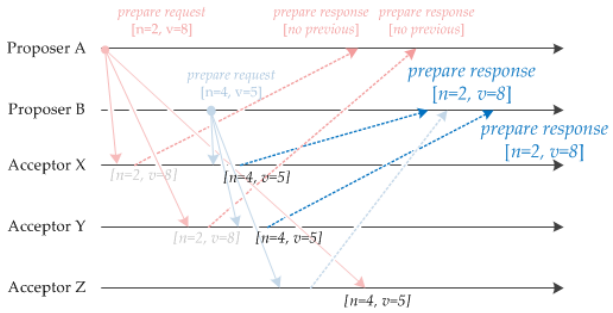

FIGURE 7 – Phase d'acceptation n°2 : prepare response.

Une fois que le proposant a reçu la préparation des réponses d'une majorité d'accepteurs, il peut émettre une demande d'acceptation. Étant donné que le proposant A n'a reçu que des réponses indiquant qu'il n'y avait pas de propositions précédentes, il envoie une demande d'acceptation à chaque accepteur avec le même numéro de proposition et la même valeur que sa proposition initiale (n = 2, v = 8). Cependant, ces demandes sont ignorées par tous les accepteurs, car toutes ont promis de ne pas

accepter les demandes avec un numéro de proposition inférieur à 4 (en réponse à la demande de préparation du proposant B).

Le proposant B envoie une demande d'acceptation à chaque accepteur contenant le numéro de proposition qu'il a utilisé précédemment (n = 4) et la valeur associée au numéro de proposition le plus élevé parmi les messages de préparation de réponse reçus (v = 8). Notons que ce n'est pas la valeur proposée initialement par le proposant B, mais la valeur la plus élevée des messages de préparation de réponse qu'il a vu. Cette phase de confirmation est illustrée figure 8.

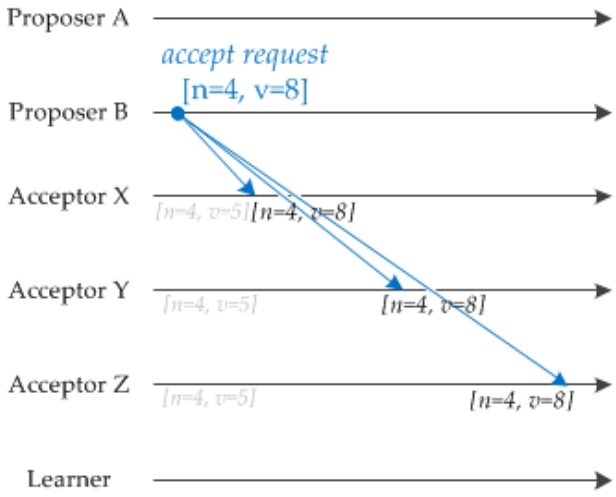

FIGURE 8 – Phase de confirmation : accept response.

Enfin, si un accepteur reçoit une demande d'acceptation pour un numéro de proposition supérieur ou égal à celui déjà vu, il accepte et envoie une notification à chaque nœud de l'apprenant. Une valeur est choisie par l'algorithme de Paxos lorsqu'un apprenant découvre qu'une majorité d'accepteurs ont accepté une valeur. Cette dernière phase est illustrée figure 9.

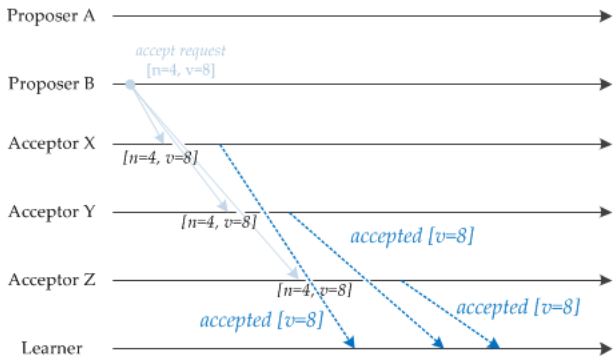

FIGURE 9 – Consensus atteint : accepted.

Une fois qu'une valeur a été choisie par Paxos, une communication ultérieure avec d'autres proposants ne peut pas modifier cette valeur. Si un autre proposant (C) envoie une demande de préparation avec un numéro de proposition supérieur à celui déjà vu, et une valeur différente (par exemple, n = 6, v = 7), chaque accepteur répond avec la proposition la plus élevée précédente (n = 4, v = 8). Cela requiert l'auteur C d'envoyer une demande d'acceptation contenant [n = 6, v = 8], ce qui confirme la valeur qui a déjà été choisie. En outre, si une minorité d'accepteurs n'a pas encore choisi de valeur, ce processus garantit qu'ils atteignent éventuellement un consensus sur la même valeur.

2.3.4 Intégration de Paxos dans Ceph

La figure 10 illustre l'intégration et la place de Paxos dans Ceph.

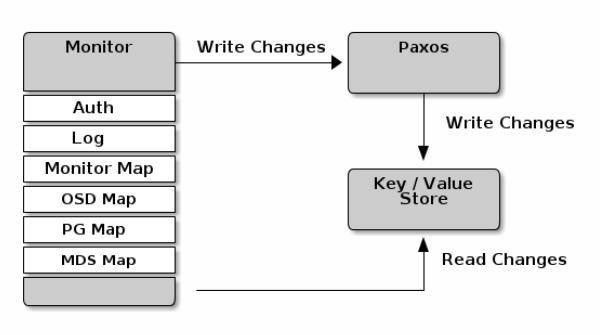

FIGURE 10 – Application de Paxos dans Ceph

Les moniteurs écrivent toutes les modifications de la carte du cluster dans une instance Paxos, qui va elle-même écrire ces modifications dans un magasin de clés/va-

leurs. Ces modifications sont ensuite exploitées par les autres moniteurs Ceph afin de mettre leur version de la carte du cluster à jour lors d'opérations de synchronisation.

2.4 CONCLUSION

Dans cette partie, nous avons étudié les concepts fondamentaux de RADOS, la solution de stockage distribuée centrale de Ceph. Nous avons vu que l'un de ses enjeux centraux est d'éliminer les points de failles uniques pouvant mettre en péril l'infrastructure.

Pour ce faire, chaque nœud du cluster de stockage prend en charge une partie de l'intelligence et de la complexité du système, à l'aide d'une version locale de l'état du cluster. Cette version est mise à jour à mesure que les composants interagissent entre eux. Le placement des données est géré par un algorithme baptisé CRUSH.

De plus, pour garantir des performances optimales, RADOS utilise un système de stockage d'objets tout en introduisant un panel de structures logiques facilitant la manipulation des données. Enfin, certains sites sont responsables de groupes d'objet et doivent assurer leur cohérence et leur pérennité au sein du cluster.

La figure 11 se propose de synthétiser l'ensemble des notions abordées dans cette partie. Elle illustre une première demande de stockage d'objets effectuée par un utilisateur de RADOS.

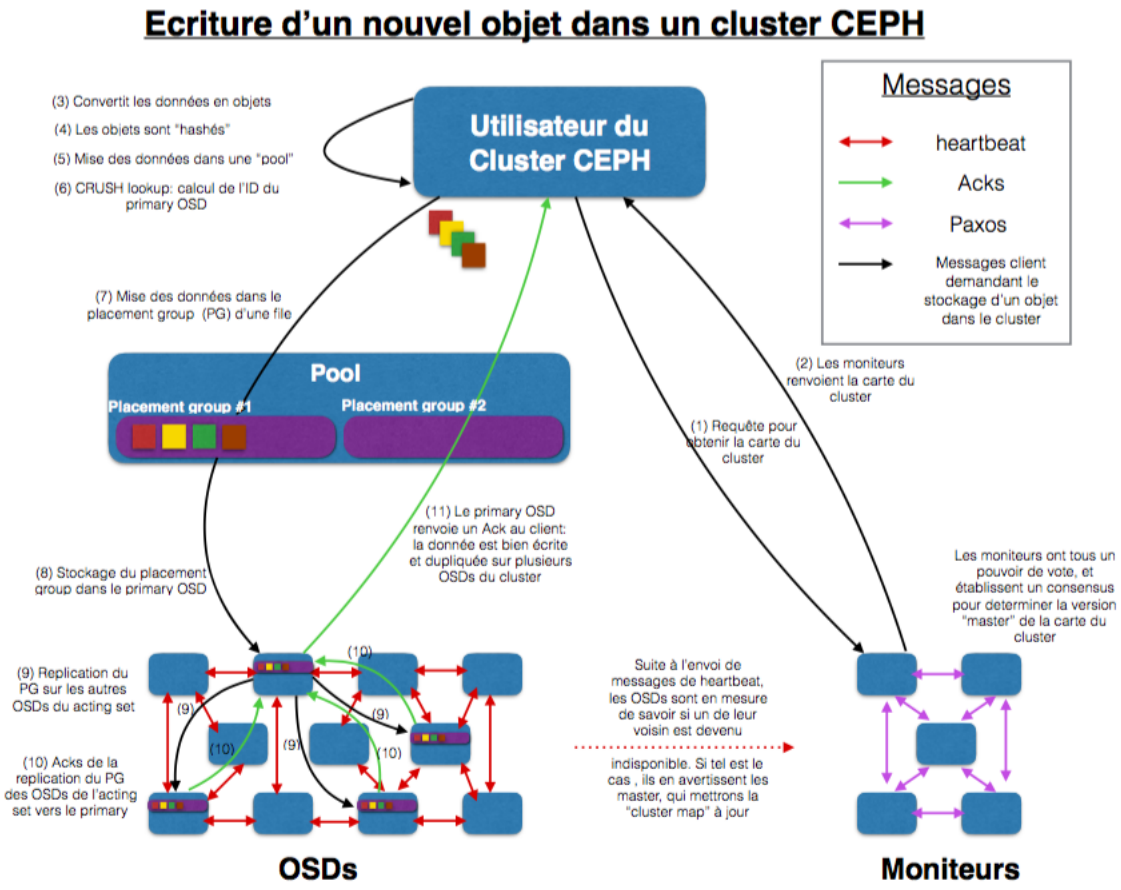

FIGURE 11 – Aperçu des différentes actions entreprise par les composants du cluster lorsqu'un utilisateur désire stocker une donnée dans le cluster.

3 CRUSH

Cette section vise est d'expliciter le fonctionnement de l'algorithme CRUSH, pierre angulaire de Ceph lui permettant de s'affranchir du principal SPOF des systèmes distribués, à savoir la **gestion du placement des objets**.

CRUSH signifie Controlled, Scalable, Decentralized Placement of Replicated Data. Comme son nom l'indique, cet algorithme vise à gérer le placement des réplicas de façon décentralisée.

3.1 PHILOSOPHIE DE CRUSH

Un des enjeux majeurs des systèmes de stockage distribué consiste d'une part à répartir de manière efficace, redondante et rapide une quantité importante de données sur plusieurs disques – des centaines voire des milliers – et d'autre part à être capable de récupérer cette information de manière tout aussi fiable, rapide et sans goulot d'étranglement.

Comme expliqué précédemment, CEPH a par ailleurs pour vocation d'être évolutif en termes de capacité de stockage (*scalable*). Ceci pose donc une contrainte d'utilisation des disques ; dans une base de données classique, les disques dernièrement ajoutés, souvent plus performants, ne sont que très peu utilisés tandis que les anciens disques ont été remplis au fur et à mesure et sont donc utilisés au maximum de leur capacités. Cette contrainte est d'autant plus forte que les vieux disques auront statistiquement plus de chance d'avoir une panne que les nouveaux.

Par ailleurs, CEPH veut s'abstraire au maximum des machines physiques pour ne pas dépendre d'un système spécifique de machines et accepter toutes sortes d'architectures.

Pour adresser ces différentes contraintes, CEPH s'appuie sur un algorithme spécifique nommé **CRUSH**. Cet algorithme est une implémentation spécifique des algo-

rithmes de type **RUSH** [1] et permet en effet d'apporter de nouvelles réponses à ces problématiques. Ainsi, CRUSH est un algorithme **pseudo-aléatoire** [2] de répartition des données, c'est-à-dire qu'il va répartir les données de manière uniforme sur l'ensemble des disques. Si la répartition des données semble donc aléatoire, CRUSH reste un algorithme **déterministe**. À contexte égal, la réponse à une entrée restera toujours la même, quel que soit le nombre d'exécutions de l'algorithme. En ce sens, CRUSH peut être *imaginé* comme la rencontre d'une fonction pseudo-aléatoire et d'une fonction de hachage.

Pour mener à bien cette répartition, CRUSH utilise une carte du cluster, ou *cluster map*. Cette carte est définie par l'administrateur et permet de spécifier l'état du système sur un grand nombre de paramètres. Elle comporte en effet une représentation sous forme d'arbre ou chaque feuille (*device*) indique le nombre de réplicats nécessaires. Chaque nœud (*bucket*) peut être lié à d'autres selon qu'il partage des points de défaillance communs, comme une même source d'alimentation ou un même réseau.

Par ailleurs, CRUSH peut mettre à jour sa carte sans réorganiser la totalité de ses données. L'algorithme répond à des objectifs de **minimisation** des déplacements inutiles des données et de scalabilité. Ainsi, lorsqu'un nouveau disque est ajouté, des données sont aléatoirement migrées sur le nouveau disque, créant un équilibre et permettant une meilleure répartition du travail dans le cluster et donc des performances accrues. Il va cependant déplacer bien moins de données qu'un algorithme de hachage classique qui nécessite bien souvent une complète réorganisation des données.

CRUSH se comporte comme une fonction de hachage, sauf lorsque la carte du cluster est modifié. Dans ce cas, certaines données sont déplacées de manière à homogénéiser le stockage, mais la plupart des données restent où elles sont. C'est ce qui le différencie d'une part de l'aléatoire et d'autre part du hachage.

1. Les algorithmes RUSH tentent de répondre à la problématique de répartition uniforme des données dans un large panel de disque pour répartir la charge d'utilisation.
2. Ce terme est employé pour des raisons de simplicité, et le caractère « pseudo-aléatoire » de CRUSH est affiné par la suite.

Trois avantages majeurs se dégagent donc :

- Chaque partie du système peut calculer l'emplacement d'un objet en ne connaissant que la carte du cluster et les règles de placement, ou *placement rules*.
- Une moindre dépendance vis-à-vis des métadonnées, CRUSH se basant plus sur des fonctions déterministes afin de déterminer l'emplacement des données à écrire ou lire.
- Une réorganisation minimale en cas de modification de l'architecture.

3.2 CARTE DU CLUSTER

Dans l'algorithme CRUSH, la distribution des données est contrôlée par une carte du cluster hiérarchique, rendant compte à la fois des ressources de stockage disponibles et des éléments logiques organisant le système.

Grâce à la cluster map, l'administrateur va pouvoir rendre compte dans l'algorithme des contraintes organisationnelles de son serveur : par exemple, il pourra tout à fait regrouper les différents disques utilisant le même réseau électrique ou internet afin d'optimiser la réplication des données sur des réseaux différents.

La cluster map est composée d'une liste de *devices* et de *buckets* : à ces deux éléments on associe un **identifiant** et un **poids**.

Les buckets permettent de définir une arborescence au sein du système et de rendre compte des contraintes matérielles de l'installation : ils contiennent des devices et peuvent contenir d'autres buckets.

Le poids est, dans le cas d'un device, assigné par l'administrateur pour contrôler le nombre de données dont il est responsable : il est en lien avec la capacité du disque. À l'inverse, dans le cas d'un bucket, le poids correspond à la somme des poids des device et des buckets qui le composent.

CRUSH se base sur quatre types de buckets différents (voir section 3.4), chacun fonctionnant avec un algorithme de sélection différent pour traiter le mouvement des données.

3.3 RÈGLES DE PLACEMENT

En rendant compte de l'architecture du système par le biais de la cluster map, CRUSH peut modéliser et, par conséquent, résoudre les éventuelles sources de défaillances liées à l'organisation générale du système.

Les sources de ces défaillances peuvent être variées : la proximité physique, une source d'alimentation partagée, réseau internet partagé etc. En encodant ces informations dans la cluster map, les placement rules vont permettre de discriminer la réplication des données sur différents domaines d'échec, tout en conservant la distribution souhaitée.

Ces règles sont définies par l'administrateur, et permettent de contrôler de manière précise la politique de réplication des données au sein du système.

3.4 LES DIFFÉRENTS TYPES DE BUCKET

D'une manière générale, CRUSH est conçu pour concilier trois objectifs concurrents : efficacité, mise à l'échelle du mapping et minimisation des opérations de migration des données lors d'un ajout ou d'un retrait d'un disque du système.

À cette fin, CRUSH définit quatre différents types de bucket, chacun impliquant l'utilisation d'une fonction différente pour le choix d'un item dans la hiérarchie (fonction $c(x, r)$ dans l'algorithme).

- **Uniform bucket** : ne peut contenir que des items de poids égaux.
- **List bucket** : choisit l'item le plus récent possible, dont le poids dépasse la somme des poids des autres items dans le bucket.
- **Tree bucket** : se rapproche dans le fonctionnement du bucket précédent, sauf qu'ici, les items sont représentés par des *arbres binaires* de recherche.
- **Straw bucket** : se base sur le jeu de la **courte paille**, en associant à chaque item du bucket une paille de longueur variable, lié au poids de l'item (un item plus lourd aura plus de chance de tirer une paille longue).

Le choix du bucket a une grande influence selon le type d'opération réalisée et la complexité de la fonction pseudo-aléatoire de sélection d'un item. Ces considérations sont résumées dans le tableau 1.

Action	Uniform	List	Tree	Straw
Complexité	O(1)	O(n)	O(log n)	O(n)
Ajout	Mauvais	Optimal	Bon	Optimal
Suppression	Mauvais	Mauvais	Bon	Optimal

TABLE 1 – Résumé de la vitesse de mapping et de l'efficacité de la migration des données suit à l'ajout ou à la suppression d'un item, pour chaque type de *bucket*.

3.5 EXPLICATION DE L'ALGORITHME

À titre indicatif, l'algorithme complet est donné dans la figure 14 de l'annexe A.

Pour stocker les données ou les retrouver – il s'agit en fait de la même opération dans le cas de CRUSH –, l'algorithme est divisé en trois étapes. L'objet en question est un argument de l'algorithme ; nous le notons x.

3.5.1 TAKE(a)

Cette fonction sélectionne un item a (en général un bucket) contenu dans la hiérarchie de la carte du cluster et l'assigne au vecteur $\vec{i}$: l'appel à cette fonction précède tout premier appel à la fonction SELECT et peut être considéré comme la constitution de la racine, en ce sens où tous les autres éléments seront inclus dans cet item.

3.5.2 SELECT(n, t)

Cette fonction permet de déterminer un nombre d'items d'un certain type à partir de la sélection préalable (voir 3.5.1). Ces items constituent l'ensemble des **candidats** pour la réplication.

En particulier, elle itère sur chaque item $i \in \vec{i}$ et choisit n éléments distincts de type t dans la sous-arborescence de la racine courante au moment de l'appel à la fonction. Les devices et buckets ont en effet un type **fixé et connu**, ce qui permet le parcours de l'arborescence et la discrimination lors du choix des items, en fonction du type passé en argument.

Pour chaque $i \in \vec{i}$, la fonction va itérer sur $r \in (1, \ldots, n)$, n correspondant au nombre d'items attendus, et va descendre récursivement dans tout bucket intermédiaire, en choisissant de manière pseudo-aléatoire un item dans chacun d'entre eux par le biais

de la fonction $c(r, x)$, propre à chaque type de bucket – jusqu'à trouver un item de type t.

Cependant, il se peut que tout ne se passe pas comme prévu, et ce pour plusieurs raisons :

- Une collision peut arriver, *e.g.* l'item choisi a déjà été choisi lors d'une précédente itération.
- Le disque choisi est marqué comme *failed* : erreur transitoire ou non.
- Le disque choisi est marqué comme *overloaded* : plus de mémoire.

Ces deux dernières erreurs sont envisageables car un disque peut être libellé comme *failed* ou *overloaded* dans la carte du cluster mais laissé dans la hiérarchie pour éviter les migrations de données inutiles.

Pour les périphériques *failed* ou *overloaded*, l'algorithme va redistribuer uniformément les éléments dans le système en redémarrant la récurrence au début de la fonction `SELECT`.

Dans le cas des collisions, un autre r' est utilisé d'abord au niveau de la hiérarchie où la collision s'est déclarée, pour tenter une recherche locale.

La fonction assigne finalement les n items distincts de type t au vecteur $\vec{i}$, qui peut soit être utilisé pour un nouvel appel à la fonction `SELECT`, soit assigné au vecteur résultat $\vec{R}$ par le biais de l'appel à la fonction `EMIT` (voir 3.5.3).

3.5.3 EMIT

Cette fonction permet de constituer le vecteur résultat $\vec{R}$ à l'aide du vecteur $\vec{i}$.

3.5.4 Gestion des erreurs

En cas d'erreur, CRUSH utilise deux systèmes de remplacement en fonction du type de réplication (**totale** ou **partielle**, *i.e.* par code correcteur).

Dans les cas de réplication primaire, *i.e.* l'objet est entièrement copié, si un des réplica subit une erreur CRUSH utilise un système dit de « *first n* ». En assignant

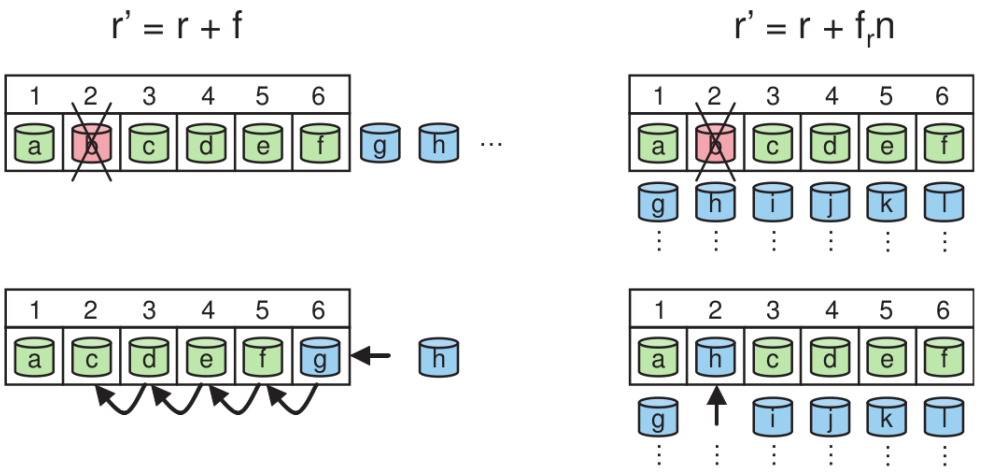

FIGURE 12 – Aperçu graphique de la gestion des erreurs dans CRUSH.

$r' = r + f$, où f est le nombre de placements échoués, les réplicas sont alors tous décalés **de** n **positions** dans la liste.

À l'inverse, dans le cas d'une réplication partielle, CRUSH considère $r' = r + f_r n$), où $f_r n$ est le nombre de tentatives échouées **sur** r. Ainsi, la séquence de candidats pour chaque réplica est indépendant des erreurs des autres.

La figure 12 illustre ce comportement.

3.6 EXEMPLE DE RÉSULTAT

Considérons une cluster map composée de *rows*, *cabinets* et de *disks* et les règles de placement indiquées dans le tableau 2.

n°	Action	Vecteur $\vec{i}$
1	take(root)	root
2	select(1, row)	row2
3	select(3, cabinet)	cab21 cab23 cab24
4	select(1, disk)	disk2107 disk2313 disk2437
5	emit	

TABLE 2 – Exemple de règles permettant d'obtenir 3 disques distincts, de trois *cabinets* différents, dans un même *row*.

1. On sélectionne la racine à partir de laquelle on va effectuer la recherche (ici *root*).
2. On sélectionne un item de type *row* (ici *row2*).
3. On sélectionne trois items de type *cabinet*, inclus dans *row2*, résultat du précédent appel à la fonction.
4. On itère sur les trois *cabinets*, et on choisi un seul *disk* contenu dans chacun d'eux.
5. On ajoute le résultat dans le vecteur $\vec{R}$.

Finalement, on se retrouve avec trois *disks*, éclatés dans trois *bucket* distincts de type *cabinet*, eux même inclus dans un *bucket* de type *row*.

La figure 13 montre une vue partielle de la hiérarchie définie par la cluster map. Les lignes en gras illustrent les items sélectionnés par chaque appel de `SELECT`.

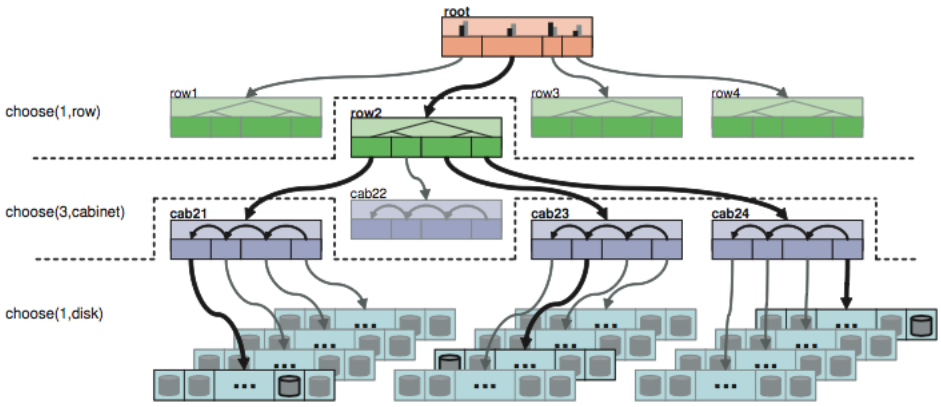

FIGURE 13 – Exemple de vue partielle de la sélection des items dans CRUSH.

4 MISE EN ŒUVRE

La vocation de ce chapitre est de fournir dans une première section, un guide d'installation permettant d'accompagner le lecteur dans la réalisation d'un premier cluster CEPH. Une seconde section se chargera de fournir et d'expliquer les diverses commandes et indicateurs démontrant la fiabilité de Ceph.

4.1 INSTALLATION ET UTILISATION

4.1.1 Architecture d'un cluster CEPH

La première étape nécessaire pour correctement préparer l'installation d'un cluster CEPH est de connaître l'architecture à réaliser.

Il existe 3 types de nœuds :

- *Object Storage Device (OSD)* : Il s'agit d'un nœud de stockage, qui s'occupe de la réplication et la reconstruction des données. Il infirme aussi le moniteur de la santé des autres OSD sur le réseau. Un minimum de **2 nœuds** est nécessaire au sein d'un cluster.
- *Moniteur* : Il maintient à jour les cartes du cluster, qui sont la carte des OSDs, la carte des « placements groups », ainsi que la carte CRUSH. Un **nombre impair de nœuds** est recommandé [1].
- *MetaData Server (MDS)* : Il sert à stocker les méta-données relatives au système de stockage CephFS. Ils ne sont donc utiles que si ce dernier est utilisé. Par ailleurs, le support des MDS étant actuellement expérimental, il ne peut y avoir qu'un seul nœud MDS.

Dans le cadre de ce tutoriel nous disposons de 6 machines. Nous avons décidé par ailleurs d'utiliser CephFS comme système de fichiers, ce qui implique l'ajout d'un nœud MDS. Le tableau 3 résume la configuration du cluster.

1. Source : http://docs.ceph.com/docs/master/rados/operations/add-or-rm-mons/ et les considérations sur les élections

Nom du nœud	Nom d'utilisateur Ceph	Type de nœud	Adresse IP
mon1	ceph07	Moniteur	172.23.2.17
mds1	ceph08	MDS	172.23.2.18
osd0	ceph09	OSD	172.23.2.19
osd1	ceph10	OSD	172.23.2.20
osd2	ceph11	OSD	172.23.2.21
osd3	ceph12	OSD	172.23.2.22

TABLE 3 – Tableau récapitulatif de l'architecture du cluster.

Une machine doit être considérée comme étant la **machine administrateur** : il s'agira de la seule machine utilisée après l'étape de mise en place du serveur SSH. Il peut s'agir de n'importe quelle machine, à la condition qu'elle soit connectée au même réseau que les autres. Lors de notre installation, ne disposant pas plus de machines, mon1 a fait office d'administrateur.

Il faut aussi prévoir des emplacements de stockage sur les OSD. Il est conseillé de prévoir des disques entiers dédiés au stockage de Ceph. Cependant, n'ayant à notre disposition qu'un unique disque par machine, nous avons décidé de dédier une partition par OSD au stockage Ceph, dans le cadre de la démonstration. Le tableau 4 récapitule la configuration de stockage de notre cluster.

Nom du nœud	Chemin de la partition	Taille de la partition
osd0	/dev/sdb4	30Go
osd1	/dev/sdb3	30Go
osd2	/dev/sdb3	30Go
osd3	/dev/sdb3	30Go

TABLE 4 – Tableau descriptif des partitions allouées au cluster Ceph par OSD.

4.1.2 Mise en place de l'infrastructure

Avant d'effectuer le déploiement de CEPH, il est nécessaire de configurer les différentes machines. Par ailleurs, il est à préciser que, dans le cas ou la machine administrateur n'est pas une machine du cluster, elle n'est pas considérée comme en faisant partie.

Installation du système d'exploitation

L'installation de Ceph se fait sur des machines disposant du noyau GNU/Linux. Nous utilisons personnellement la distribution *Debian 8 (Jessie)*, toutefois le choix de celle-ci reste libre : d'autres distributions telles que CentOS et Ubuntu-Server peuvent être utilisées.

Du fait que nous utilisons *Debian 8*, il est nécessaire de mettre à jour le noyau dans une version plus récente (4.X ou plus). La procédure de mise à jour du noyau est détaillée à l'annexe B.

Pour plus d'informations sur les prérequis spécifiques à d'autres distributions, il est nécessaire de se référer à la page « OS Recommendations » de la documentation officielle [2].

Configuration des interfaces réseau (NIC)

Le cluster CEPH nécessite que chaque nœud du cluster puisse communiquer avec les autres. Il faut donc s'assurer que chacune des machines possède une adresse unique sur le réseau.

La configuration est à appliquer **sur chaque machine du cluster** se trouve dans le fichier `/etc/networks/interfaces`. Le code 1 montre cette configuration.

```
auto enp4s0
iface enp4s0 inet static
address 192.168.0.10
netmask 255.255.255.0
network 192.168.0.0
broadcast 192.168.0.255
```

2. Liste des systèmes d'exploitation recommandés : http://docs.ceph.com/docs/master/start/os-recommendations/

Listing 1 – Configuration des machines du cluster.

Configuration des noms d'hôte

Chaque machine du cluster doit pouvoir contacter les autres machines via un nom. Par ailleurs la machine administrateur n'a pas besoin d'être contactée mais doit connaître le nom de l'ensemble des machines. Pour permettre cela, nous allons éditer le fichier `/etc/hosts` **sur chaque machine**.

Il est nécessaire d'éditer le fichier avec les droits root.

Les lignes ajoutées à la fin du fichier, pour notre cluster, sont visibles dans le code 2.

```
#Moniteur IP
172.23.2.17mon1

#MDS IP
172.23.2.18mds1

#OSD IPs
172.23.2.19osd0
172.23.2.20osd1
172.23.2.21osd2
172.23.2.22osd3
```

Listing 2 – Modification du fichier hosts.

Configuration des utilisateurs Ceph

Pour le déploiement, la machine administrateur, via l'utilitaire `ceph-deploy` va devoir accéder aux différentes machines sans utiliser de mot de passe.

Nous allons donc créer des utilisateurs pour chaque machine du cluster, à qui nous allons donner les accès à l'utilisateur root sans mot de passe.

L'ensemble des étapes de relatives à la mise en place d'un utilisateur doivent donc être appliquées **sur chaque machine du cluster**. Aussi, le choix des noms d'utilisateur est libre, dans notre cas nous avons pris ceux présentés dans le tableau 3.

Avant toute chose, il faut installer le paquet `sudo`. Il permet, dans sa configuration, de donner un accès root sans mot de passe à un utilisateur.

```
$ su
$ apt-get install sudo
$ exit
```

La commande ci-dessous permet de créer un nouvel utilisateur, qui n'aura, par la suite, pas de mot de passe root.

Il est possible d'utiliser l'utilisateur courant, au lieu de créer un nouvel utilisateur. Il faut être conscient que dans ce cas, l'utilisateur courant aura un accès root via la commande sudo sans mot de passe.

```
$ sudo useradd -d /home/<nom d'utilisateur> -m <nom d'utilisateur>
$ sudo passwd <nom d'utilisateur>
```

Enfin, nous donnons un accès sudo sans mot de passe à notre utilisateur :

```
$ echo "<nom d'utilisateur> ALL = (root) NOPASSWD:ALL" | sudo tee /etc/sudoers.d/<nom d'
    utilisateur>
$ sudo chmod 0440/etc/sudoers.d/<nom d'utilisateur>
```

Configuration SSH

Afin de permettre à `ceph-deploy` d'accéder aux nœuds via SSH, il faut installer le paquet `openssh-server` **sur chaque machine**, si cela n'est pas déjà fait :

```
$ su
$ apt-get install openssh-server
$ exit
```

Nous générons ensuite un jeu de clés SSH (privées/publiques) **depuis la machine administrateur**, dont nous transmettrons ensuite la clé publique à tous les nœuds afin de pouvoir s'y connecter sans mot de passe.

> Il est préférable de laisser l'ensemble des champs vides, il suffit donc d'appuyer sur entrée à chaque question.

```
$ ssh-keygen
```

Voici la commande à exécuter **depuis la machine administrateur**, pour transmettre la clé SSH publique **à chacun des nœuds du cluster** :

```
$ ssh-copy-id <nom d-utilisateur>@<nom du noeud>
```

Dans notre cas cela donne :

```
$ ssh-copy-id ceph07@mon1
$ ssh-copy-id ceph08@mds1
$ ssh-copy-id ceph09@osd0
$ ssh-copy-id ceph10@osd1
$ ssh-copy-id ceph11@osd2
$ ssh-copy-id ceph12@osd3
```

La dernière étape de la configuration SSH est la création du fichier `\~/.ssh/config` **sur l'ensemble des machines**, qu'il faut remplir comme montré dans le code 3.

```
Host mon1
  Hostname mon1
   User  ceph07
Host mds1
  Hostname mds1
   User  ceph08
Host osd0
  Hostname osd1
   User  ceph09
Host osd1
  Hostname osd1
   User  ceph10
```

```
Host osd2
  Hostname osd2
   User  ceph11
Host osd3
  Hostname osd3
   User  ceph12
```

Listing 3 – Configuration SSH.

Puis il faut rendre le fichier non éditable, sauf pour l'utilisateur root :

```
$ su
$ chmod 600~./ssh/config
$ exit
```

Configuration du serveur NTP (Synchronisation de l'horloge)

Afin de garantir que horloges des nœuds soient synchronisées entre elles, il faut installer le paquet ntp.

```
$ sudo apt-get install ntp
```

4.1.3 Installation de CEPH et mise en route

Dans cette partie, il va être question dans une première partie d'installer le paquet `ceph-deploy`, permettant par la suite de déployer Ceph sur le cluster.

L'intégralité des commandes qui vont suivre seront exécutées **sur la machine administrateur**.

Installation du paquet ceph-deploy

Il faut tout d'abord ajouter le repository de Ceph aux sources du gestionnaire de paquet, puis mettre à jour la base de donnée de référencement des paquets, et enfin lancer l'installation de Ceph, le tout grâce aux commandes suivantes du code 4.

```
$ wget -q -O- 'http://eu.ceph.com/keys/release.asc' | sudo apt-key add -
$ echo deb http://eu.ceph.com/debian-kraken/ $(lsb_release -sc) main | sudo tee /etc/apt/
    sources.list.d/ceph.list
$ sudo apt-get update && sudo apt-get install ceph-deploy
```

Listing 4 – Installation du paquet ceph-deploy

Déploiement

Il faut tout d'abord créer un dossier d'administration qui contiendra, entre autre, la configuration de notre cluster CEPH :

```
$ mkdir ceph-cluster
$ cd ceph-cluster
```

Nous exécutons ensuite la commande permettant de déclarer les moniteurs initiaux, il sont ainsi enregistré dans la configuration du cluster Ceph (`ceph.conf`).

Remarque : Il est possible de ne déclarer que le moniteur initial, et d'en ajouter d'autres par la suite.

```
$ ceph-deploy new mon1
```

Dans le cas ou le nombre d'OSD est inférieur à 3, il faut modifier dans le fichier `ceph.conf`, tel qu'indiqué dans le code 5.

```
osd pool default size = 3
osd pool default min size = 2
osd crush chooseleaf type = 1# Multi-node cluster in a single rack
```

Listing 5 – Congifuration de Ceph dans le cas d'un faible nombre d'OSD.

On installe ensuite Ceph sur l'ensemble des machines, à part le nœud MDS. Il est important d'y spécifier le nœud administrateur pour qu'il bénéficie des commandes nécéssaires pour administrer le cluster.

```
$ ceph-deploy install mon1 osd0 osd1 osd2 osd3
```

On ajoute ensuite les moniteurs initiaux (déclarés précédemment) dans le cluster, afin qu'ils forme un quorum et que les fichiers de clés puissent être générés. Plusieurs fichiers au format `.keyring` sont donc normalement dans le dossier d'administration.

```
$ ceph-deploy mon create initial
```

Le cluster est donc normalement créé. Néanmoins, nous n'avons pas encore indiqué les partitions de stockage des OSDs. Voici la commande à éxecuter :

```
$ ceph-deploy osd prepare --fs-type btrfs {ceph-node}:/path/to/directory
$ ceph-deploy osd activate {ceph-node}:/path/to/directory
```

Dans notre configuration, cela donne pour l'OSD 0 :

```
$ ceph-deploy osd prepare --fs-type btrfs osd0:/dev/sdb4
$ ceph-deploy osd activate osd4:/dev/sdb4
```

Pour pouvoir exécuter des commandes administrateur sur l'ensemble des sites, sans préciser la clé d'administration, ni le nom du site administrateur :

Dans notre configuration, il s'agit de la commande suivante :

```
$ ceph-deploy admin mon1 osd0 osd1 osd2 osd3
```

Cette commande ajoute, entre autre le fichier `ceph.client.admin.keyring` dans le dossier `/etc/` de chaque machine, ce fichier contient la clé d'administrateur. La commande ne configurant pas correctement les droits du fichier, il faut donc lancer la commande suivante **sur chaque machine**, elle donne un accès en lecture à l'utilisateur root :

```
$ sudo chmod +r /etc/ceph/ceph.client.admin.keyring
```

On peut ensuite vérifier l'état du cluster, si tout les paramétres sont bon la commande suivante doit retourner `HEALTH_OK`.

```
$ ceph health
```

Mise en place de CephFS

Dans cette partie, l'objectif est de mettre en place CephFS, afin de pouvoir déposer/récupérer des données dans le cluster Ceph, via un point de montage réseau. Pour rappel, notre nœud MDS s'appelle mds1. Les commandes du code 6 vont, dans l'ordre, installer ceph sur la machine, définir le nœud comme étant un MDS, lui donner les droits d'administration, et enfin se connecter en SSH pour donner les droits en lecture sur le fichier de clé d'administration.

```
$ ceph-deploy install mds1
$ ceph-deploy mds create mds1
$ ceph-deploy admin mds1
$ ssh mds1
$ sudo chmod +r /etc/ceph/ceph.client.admin.keyring
```

Listing 6 – Installation et mise en place de Ceph pour un MDS

La création de CephFS nécéssite la création de deux pools, une première sera appelée `cephfsdatapool` et sert aux stockage des données, une seconde nommée `cepfsmetadatapool` sert au stockage des métadonnées. La dernière commande permet le lancement du service CephFS. Le code 7 montre ces commandes.

```
$ ceph osd pool create cephfsdatapool 128128
$ ceph osd pool create metadatapool 128128
$ ceph fs new cephfs0 cephfsmetadatapool cephfsdatapool
$ ceph fs ls // verifier que tout s'est bien passe
```

Listing 7 – Création des pools

Avant de pouvoir utiliser CephFS, il faut créer un point de montage à partir duquel on pourra y accéder.

```
$ sudo mkdir /mnt/cephfs
```

Ensuite, on récupère la clé du nœud administrateur (`mon1`), que l'on va placer dans un fichier `admin.secret`. Pour cela on utilise une regex sur le fichier de clé pour la récupérer.

```
$ cat /etc/ceph/ceph.client.admin.keyring | grep key | grep -o "[^$(printf '\t key = ')
    ].*" > ~/ceph-cluster/admin.secret
```

On peut ensuite monter CephFS sur le dossier créé plus haut. L'adresse utilisée est celle du moniteur, le port est celui de cephfs par défaut.

```
$ sudo mount -t ceph 172.23.2.17:6789:/ /mnt/cephfs -o name=mon1,secretfile=~/ceph-
    cluster/admin.secret
```

Pour utiliser CephFS, il faut utiliser le dossier présent dans `/mnt/cephfs`. Pour valider le fonctionnement, il suffit de surveiller l'action du cluster après le dépôt d'un fichier.

```
$ ceph -w
$ sudo mv /path/to/some/file /mnt/cephfs/
```

Autres commandes

Il est par exemple possible de créer de nouvelles pools via la commande :

```
$ ceph osd pool create <nom de la pool> <nombre de placement groups> <idem>
```

Voici un exemple de commande de création de pool :

```
$ ceph osd pool create poule 128128
```

Par défaut la pool crée est « en mode » réplication, il est aussi possible de configurer la pool pour gérer les codes correcteurs d'erreurs. Il est également possible d'ajouter un objet manuellement dans une pool, comme suit :

```
$ rados -p <nom de la pool> put <nom de l'objet> <nom du fichier>
$ ceph osd map <nom de la pool> <nom de l'objet>
```

4.2 REPRISE SUR ERREUR ET TOLÉRANCE AUX PANNES

Dans cette section, nous montrons un cas d'exemple où Ceph est capable de mitiger les erreurs produites.

4.2.1 Outils de monitoring de Ceph

Monitoring des OSD

On peut visualiser l'état des OSD du cluster comme suit :

```
$ ceph osd tree
$ ceph health [detail]
```

On a donc accès aux informations suivantes :

- *Nombre d'OSD up* : Les OSD actifs qui ont été déclaré au(x) moniteur(s) ;
- *Nombre d'OSD down* : les OSD considérés non fonctionnel dans le cluster qui ont été déclarés au(x) moniteur(s) ;
- *Nombre d'OSD in* : Les OSD inclus dans le cluster (peuvent être non fonctionnel) ;
- *Nombre d'OSD out* : Les OSD en dehors du cluster (peuvent être fonctionnel) ;
- *Weight* : L'importance d'un OSD dans le cluster (entre 0 et 1, par défaut a 1).

En pratique, le cas problématique est celui d'un OSD **in** et **down**. En d'autres termes, l'OSD est présent dans le cluster mais non fonctionnel. Le cas typique est un OSD non fonctionnel suite a un problème matériel. Le cas de la déconnexion d'un OSD sera traitée dans la section 4.2.2.

Monitoring des pools

Pour lister les pools, on peut utiliser la commande suivante :

```
$ ceph osd lspools
```

Pour lister les objets existants dans une pool, on peut utiliser la commande suivante :

```
$ rados -p {pool name} ls
```

Monitoring des données

Pour afficher en temps réel le peering des données, on peut utiliser la commande suivante :

```
$ ceph -w
```

Et pour connaître la localisation d'un objet dans une pool, *i.e.* quels OSD le prennent en charge :

```
$ ceph osd map {poolname} {filename}
```

4.2.2 Déconnexion d'un osd

Déconnexion et reconnexion manuelle

Dans le but de réaliser une maintenance, il peut être nécessaire de retirer manuellement un OSD du cluster (OSD up et out) puis de l'arrêter pour pouvoir travailler dessus.

```
$ ceph osd out {osd-num} #force the osd out
$ sudo /etc/init.d/ceph stop osd.{osd-num} #stop osd
```

Dans ce cas, la crush map sera recalculée car l'état du cluster n'est plus le même (*OSD out*). Lors d'une telle manipulation, il faut impérativement vérifier que le cluster dispose de d'un volume de stockage suffisant pour compenser la perte temporaire d'un OSD.

Après la maintenance, il faut remettre l'OSD en place, pour cela il faut le redémarrer puis le replacer dans le cluster.

```
$ sudo start ceph-osd id={osd-num} #restart osd from monitor
$ ceph osd in {osd-num} #force the osd in
```

Dans le cas ou le volume des données est important, il peut être judicieux d'intégrer l'OSD au fur et a mesure. En effet, il est possible de choisir le poids d'un OSD dans

la crush map. Cela permet d'augmenter son importance dans le cluster au fur et a mesure de son intégration.

```
$ ceph osd crush reweight {osd-id} .{poids entre 0et 1.0}
```

Il est possible de visualiser le poids (*weight*) des OSD via la commande suivante :

```
$ ceph osd tree
```

Déconnexion accidentelle

À un moment indéterminé, il est possible qu'un OSD tombe en panne ou se retrouve déconnecté du réseau. Il peut être alors nécessaire d'utiliser les outils de monitoring pour évaluer l'état du cluster. Dans le cas ou un OSD est hors-service (*i.e. down*), il y aura un temps de latence avant que le cluster ne le considère *out*.

Pour configurer ce temps de latence (sachant qu'un intervalle de o empêchera l'OSD d'être marqué *out*), il est possible d'utiliser la commande suivante :

```
$ mon osd down out interval = {delay}
```

Pour avoir des informations sur une erreur, il est possible de vérifier le réseau ou les logs de la machine :

```
$ cat /var/log/ceph/{cluster-name}.log
$ netstat -s
```

Pour relancer un OSD, la procédure est la même que s'il avait été enlevé manuellement ; néanmoins, il faut prendre soin de ne pas réintégrer un OSD défectueux dans le cluster. À ce titre, il est nécessaire de savoir pourquoi il en a été exclu en premier lieu. Le code 8 illustre cette manipulation.

```
ceph osd set noout #permet d'empecher l'osd d'etre marque out (maintenance /flappy disk)
ceph osd unset noout #suppr commande precedente

ceph osd set noup # prevent OSDs from getting marked up
ceph osd set nodown # prevent OSDs from getting marked down
```

```
ceph osd unset noup
ceph osd unset nodown
```

Listing 8 – Procédure de relancement d'un OSD.

Problèmes possibles

Des problèmes de connexion ou de lancement du démon `ceph` peuvent parfois apparaître. Cela peut provoquer un drapeau *out* ou *down* sur un OSD pourtant parfaitement fonctionnel.

La maintenance sur un ou plusieurs OSD peut aussi poser des risques. Pour éviter que cela implique la recréation de la crush map, il peut être nécessaire d'utiliser une des commandes suivantes :

```
ceph osd set noout
ceph osd set noin
ceph osd set noup
ceph osd set nodown
```

Ces commandes empêchent respectivement l'application du flag *out*, *in*, *up* et *down*. Cela veut dire qu'il est possible de forcer le système à considérer un OSD comme étant *up* et *in* même si celui-ci a été retiré.

Ces commandes sont là pour outrepasser les comportements automatiques de Ceph ; à ce titre, il est recommandé de les exécuter avec prudence car il est alors nécessaire de défaire ces drapeaux manuellement. Pour ce faire, il faut utiliser la commande `unset` :

```
ceph osd unset noout
ceph osd unset noin
ceph osd unset noup
ceph osd unset nodown
```

5 CONCLUSION

À l'heure où des données sont produites chaque instant en proportions considérables, l'étude de Ceph comme solution de stockage distribué nous a permis de mesurer l'enjeu des systèmes distribués. Alors qu'il existe un large panel de solutions de stockage, Ceph se distingue de ses concurrents en adoptant une approche novatrice. Sa philosophie a comme fondement l'élimination de tous les points de défaillance uniques et de toute composante qui pourrait constituer un goulot d'étranglement. Aussi, l'idée est de s'approcher au maximum d'un système uniforme où le comportement est semblable d'une machine à une autre. Ceph a pour socle RADOS, composé d'un de nœuds de stockage et de nœuds moniteurs. Ce système distribué prend en charge la distribution des données au travers du cluster de machines. L'algorithme CRUSH permet de distribuer les données de façon homogène et décentralisée.

L'étude de Ceph nous a permis de prendre du recul sur les notions étudiées en SR05 ce semestre. Ainsi, nous avons pu observer la manière avec laquelle étaient traitées les problématiques de distribution de données, la gestion du temps entre les machines, ainsi que les élections et la reprise sur erreurs.

Finalement, la phase d'étude s'est achevée par le déploiement d'un cluster Ceph. L'infrastructure mise en place vise à démontrer le potentiel de Ceph, et a été pour nous l'occasion d'observer effectivement les comportements étudiés. Cette étape s'inscrit dans la continuité de l'étude théorique et conclut parfaitement ces semaines de travail.

De nos jours, des objets « *connectés* » sont produits et installés massivement aux quatre coins du monde. Au-delà des questions éthiques que cela soulève, il est pertinent de s'interroger sur les méthodes de stockage de telles quantités de données. Les systèmes de stockage actuels vont rencontrer des limites matérielles et devront nécessairement s'adapter pour répondre aux contraintes modernes. Bien que Ceph parvienne à se distinguer de ses concurrents, certains aspects, dont la communication client/cluster, la sécurité et le contrôle des accès doivent être améliorés pour faire de cette solution open-source une solution entièrement aboutie et, peut-être, dominer le marché du stockage distribué.

A ALGORITHME CRUSH

Algorithm 1 CRUSH placement for object x

```
1: procedure TAKE(a)                          ▷ Put item a in working vector i⃗
2:     i⃗ ← [a]
3: end procedure

4: procedure SELECT(n,t)                      ▷ Select n items of type t
5:     o⃗ ← ∅                                  ▷ Our output, initially empty
6:     for i ∈ i⃗ do                           ▷ Loop over input i⃗
7:         f ← 0                              ▷ No failures yet
8:         for r ← 1,n do                     ▷ Loop over n replicas
9:             f_r ← 0                        ▷ No failures on this replica
10:            retry_descent ← false
11:            repeat
12:                b ← bucket(i)              ▷ Start descent at bucket i
13:                retry_bucket ← false
14:                repeat
15:                    if "first n" then
16:                        r' ← r + f
17:                    else
18:                        r' ← r + f_r n
19:                    end if
20:                    o ← b.c(r',x)
21:                    if type(o) ≠ t then
22:                        b ← bucket(o)      ▷ Continue descent
23:                        retry_bucket ← true
24:                    else if o ∈ o⃗ or failed(o) or overload(o,x)
   then
25:                        f_r ← f_r + 1, f ← f + 1
26:                        if o ∈ o⃗ and f_r < 3 then
27:                            retry_bucket ← true      ▷ Retry
   collisions locally (see Section 3.2.1)
28:                        else
29:                            retry_descent ← true     ▷ Otherwise
   retry descent from i
30:                        end if
31:                    end if
32:                until ¬retry_bucket
33:            until ¬retry_descent
34:            o⃗ ← [o⃗,o]                      ▷ Add o to output o⃗
35:        end for
36:    end for
37:    i⃗ ← o⃗                                  ▷ Copy output back into i⃗
38: end procedure

39: procedure EMIT                            ▷ Append working vector i⃗ to result
40:    R⃗ ← [R⃗,i⃗]
41: end procedure
```

FIGURE 14 – Détail de l'algorithme CRUSH.

B MISE À JOUR DU NOYAU - DEBIAN 8

La distribution Debian 8 (Jessie) intègre par défaut une version 3.X du noyau. Dans le cadre de notre installation, nous avons choisi d'utiliser btrfs comme système de fichiers des partitions de stockage Ceph des OSDs, or ce système de fichier est stable depuis les versions 4.X du noyau.

Les commande suivantes permettent de mettre à jour le noyau vers la version stable la plus récente, dans notre cas la 4.9 :

```
$ sudo echo deb http://http.debian.net/debian jessie-backports main > /etc/apt/sources.
    list.d/jessie-backports.list
$ sudo apt-get update
$ sudo apt-get -t jessie-backports install linux-image-amd64
$ sudo apt-get -t jessie-backports install linux-headers-amd64
```

TABLE DES FIGURES

LISTE DES TABLEAUX

BIBLIOGRAPHIE

[1] Chrisophe Bardy. *Comprendre les bases de l'architecture de Ceph*. Juin 2016. URL : http://www.lemagit.fr/conseil/Comprendre-les-bases-de-larchitecture-de-Ceph.

[2] *Ceph Essentials*. 30 mai 2014. URL : http://storageconference.us/2014/Presentations/Tutorial-CEPH.pdf.

[3] Olivier Delhomme. « Presentation et installation du systeme de stockage reparti Ceph ». In : *Linux Magazine* 179 (2015), p. 32.

[4] Olivier Delhomme. « Utilisez Ceph, le systeme de fichiers distribue haute performance ». In : *Linux Magazine* 180 (2015), p. 30.

[5] *Documentation*. 2016. URL : http://docs.ceph.com/docs/master/.

[6] Alan Johnson. *Ceph – Hands-on guide*. URL : https://alanxelsys.com/ceph-hands-on-guide/.

[7] John F. Kim. *Ceph Is A Hot Storage Solution – But Why ?* URL : http://www.mellanox.com/blog/2015/06/ceph-is-a-hot-storage-solution-but-why/.

[8] Karan Singh. *Learning Ceph*. Packt Publishing Ltd, 2015.

[9] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Darrell de Long et Carlos Maltzahn. « Ceph : A scalable, high-performance distributed file system ». In : *Proceedings of the 7th symposium on Operating systems design and implementation*. USENIX Association. 2006, p. 307–320.

[10] Sage A. Weil, Andrew W. Leung, Scott A. Brandt et Carlos Maltzahn. « RADOS : A Scalable, Reliable Storage Service for Petabyte-scale Storage Clusters ». In : *Proceedings of the 2Nd International Workshop on Petascale Data Storage : Held in Conjunction with Supercomputing '07*. PDSW '07. Reno, Nevada : ACM, 2007, p. 35–44. ISBN : 978-1-59593-899-2. DOI : 10.1145/1374596.1374606. URL : https://ceph.com/wp-content/uploads/2016/08/weil-rados-pdsw07.pdf.