ÉVALUATION D'APACHE OZONE ET DE SON INTÉGRATION DU PROTOCOLE DE CONSENSUS RAFT AVEC LA LIBRAIRIE APACHE RATIS AFIN D'OBTENIR UN SYSTÈME DISTRIBUÉ FIABLE ET COHÉRENT

MÉMOIRE PRÉSENTÉ COMME EXIGENCE PARTIELLE DE LA MAÎTRISE EN TECHNOLOGIE DE L'INFORMATION

PAR

NICOLAS MATHON

DÉCEMBRE 2024

RÉSUMÉ

De nos jours, l'utilisation de systèmes distribués est monnaie courante. Traditionnellement, l'entreprise utilisait l'entrepôt de données (*data warehouse*), lequel utilisait la méthodologie « *schema on write* » pour emmagasiner l'information dans une structure de données robuste et optimisée. Par contre, cette rigidité la rendait difficile à modifier comparativement à celle du « *schema on read* » (Sebaa et al., 2017), telle qu'utilisée dans les lacs de données (*data lake*), terme inventé en 2010 par James Dixon, fondateur de la compagnie Pentaho. Ce dernier représente un environnement distribué qui permet de stocker et traiter une quantité phénoménale de données brutes non structurées, semi-structurées ou structurées. Ces dernières sont ensuite ingérées massivement et rapidement sans aucun processus de transformation intermédiaire (*Extract, Transform and Load*) car différentes structures de données seront appliquées uniquement lorsqu'elles seront utilisées (*Extract, Load and Transform*), pour y effectuer des analyses qui permettront à l'organisation de prendre des décisions éclairées pour assurer sa pérennité (Madera & Laurent, 2016). Relativement peu coûteux, ils permettent de relier plusieurs machines entre elles afin de former une grappe (*cluster*) qui permettra notamment de répartir la charge et de répliquer les données afin d'obtenir un environnement doté de haute disponibilité (Nachiappan, 2020).

L'un des problèmes communs est de conserver la cohérence entre les réplications lors d'erreurs ou d'échecs (J.-S. Ahn et al., 2019). Parmi ces systèmes distribués, nous retrouvons un récent projet débuté en 2018 par la Apache Software Foundation (ASF) et nommé Apache Ozone (Ozone), dont la première version officiellement disponible au grand public est parue en septembre 2020 (Ozone, 2023). Ce produit possède plusieurs particularités dont celle de pouvoir gérer efficacement les petits fichiers, contrairement à Apache Hadoop (Hadoop) pour lequel c'est une problématique (Bende & Shedge, 2016), tout en étant hautement extensible (Mehmood et al., 2024). De plus, il intègre un autre logiciel de ASF, Apache Ratis (Ratis), qui est une implémentation Java du protocole de consensus Replicated And Fault Tolerant (RAFT), similaire à Paxos et Viewstamped Replication, basés eux aussi sur l'élection d'un *leader* (Liu, 2018), permettant la tolérance aux pannes grâce à ses processus de validation, d'entente et de répli-

cation d'une même valeur entre toutes les machines (Ongaro & Ousterhout, 2014), offrant ainsi une excellente redondance. Le choix de RAFT n'est probablement pas le fruit du hasard, car ce dernier a su se forger une solide communauté ayant développé des versions dans une panoplie de langages notamment C++, Python et Java (Howard, 2014) et il est utilisé dans différents systèmes tels que etcd, CockroachDB et neo4j pour ne nommer que ceux-ci. Une chose est certaine, c'est que les pannes font déjà partie du quotidien et que les systèmes doivent être en mesure de les gérer tout en demeurant cohérents.

Notre projet de recherche consiste donc à bâtir une infrastructure Ozone afin de pouvoir vérifier la fiabilité et mesurer la cohérence entre les différents noeuds (*nodes*) composant ce que nous appelons le RAFT *cluster*. Nous allons utiliser une méthodologie basée sur l'expérience et les simulations afin de produire nos résultats et valider notre hypothèse.

ABSTRACT

Nowadays, distributed systems are commonly used. Traditionally, the enterprise used the data warehouse, which employed the « *schema on write* » methodology to store information in a robust and optimized data structure. This rigidity made it really difficult to modify compared to the « *schema on read* » methodology (Sebaa et al., 2017), as used in data lakes, a term coined in 2010 by James Dixon, founder of the Pentaho company. The latter represents a distributed environment for storing and processing phenomenal amounts of unstructured, semi-structured or structured data, which can be ingested massively and rapidly without any intermediate process (*Extract, Transform and Load*), as different data structures are applied only when they are used (*Extract, Load and Transform*), to perform analyses that will help the organization to make informed decisions to ensure its sustainability (Madera & Laurent, 2016). Relatively inexpensive, they can be used to link several machines together to form a cluster, enabling load balancing and data replication to create a high-availability environment (Nachiappan, 2020).

A common problem is maintaining consistency between the replicas during errors or failures (J.-S. Ahn et al., 2019). Among these systems, we find a recent project started in 2018 by the ASF called Ozone, whose first official version available to the general public was released in September 2020 (Ozone, 2023). This product has a number of special features especially that it can efficiently handle small files, unlike Hadoop that has this problematic (Bende & Shedge, 2016). Also, it integrates another ASF product, Ratis, which is a Java implementation of the RAFT consensus protocol, similar to Paxos and Viewstamped Replication algorithms (Liu, 2018), which are also leader-based systems enabling fault tolerance in its processes for validating, agreeing and replicating a same value between all the machines (Ongaro & Ousterhout, 2014). The choice of RAFT is probably not a coincidence, as it has built up a strong community that has developed versions in a wide range of languages including C++, Python and Java (Howard, 2014) and it is used in various systems such as etcd, CockroachDB and neo4j to name only a few. One certain thing, failures are already part of everyday life, and systems must be able to handle them while remaining consistent.

Our research project therefore consists in building an Ozone infrastructure in order to verify the reliability and to measure the consistency between the different *nodes* making up the RAFT cluster. We use a methodology based on experimentations and simulations to produce our results and validate our hypothesis.

REMERCIEMENTS

Tout d'abord, j'aimerais remercier M. Daniel Lemire pour son soutien et sa grande disponibilité tout au long de mon parcours de maîtrise, qui fût parsemé de doutes et d'embûches. Son écoute, sa rigueur et son professionnalisme m'ont réellement poussé à me surpasser et ont grandement contribué à la réalisation de mes travaux.

Dans un deuxième temps, je ne peux passer sous silence l'appui de mon employeur, l'École nationale de police du Québec (ENPQ), qui encourage ses employés à repousser leurs limites en poursuivant leur formation académique grâce à son support technologique, d'aménagement de temps de travail et financier. Je remercie plus particulièrement mes supérieurs Nathalie Bournival et Patrick Lefebvre qui m'ont donné leur confiance et m'ont permis d'utiliser les ressources logicielles, matérielles et humaines requises pendant tout mon cheminement. Je remercie aussi mes collègues pour leur compréhension envers mes multiples absences afin de travailler mon mémoire.

Merci à la communauté Ozone notamment Ethan Rose, Arpit Agarwal et Siddhant Sangwan qui m'ont, à plusieurs reprises, offert leur aide et conseils via le canal Slack #Ozone.

Merci à mon grand ami Christian Bellemare pour ses précieux conseils liés à la recherche, à LaTeXet R Project.

Enfin, il est très important pour moi de terminer en mentionnant l'énorme appui de ma famille immédiate, mon garçon Isaac et ma conjointe Louise Nollet. Ils ont su accepter les nombreuses heures à travailler sur mon projet de recherche ainsi qu'à la rédaction de ce mémoire et ont su m'épauler dans les moments les plus difficiles notamment pendant la pandémie.

Merci à Christiane Lesage et Johanne Gaudreau qui m'ont aidé de l'au-delà.

Nicolas Mathon
Saint-Boniface-de-Shawinigan
2 décembre 2024

Pour Isaac,

*Refais chaque jour le serment d'être heureux (*Alain, *1985).*

TABLE DES MATIÈRES

Liste des tableaux

Table des figures

Liste des extraits de code

Abréviations, sigles et acronymes

A

ACID Ensemble des propriétés (*Atomic, Consistent, Isolated & Durable*) qui permettent de garantir qu'une commande ou transaction informatique s'est exécutée de façon fiable. 40

ASF Apache Software Foundation est une organisation à but non lucratif qui développe des logiciels ouverts (*open-source*). i, iii, 13, 27

C

CRUD Opérations de base *Create, Read, Update & Delete* pour la persistance des données. 32, 40, 41, 52, 53, 78

D

DN *Data Node*, un type de *node* pour stocker des données et améliorer la résilience du système. 16, 18, 21, 23–25, 46, 55, 58, 67

E

EC Fonctionnalité de réplication par *erasure coding* et qui réduit l'utilisation de stockage de 50% comparativement à RAFT. 16, 18, 23

ENPQ École nationale de police du Québec, institution québécoise dont la mission principale est d'assurer la formation de tous les policiers actifs sur son territoire. v, 29, 34, 36, 37

H

HA Habileté d'un système à opérer en continu sans intervention et lui assurant sa disponibilité. 15, 80

Hadoop Apache Hadoop est un système distribué pour le stockage et le traitement de fichiers volumineux. i, iii, xii, 3, 12–14, 21, 22, 26, 27, 30

Hive Apache Hive est un logiciel d'entrepôt de données reposant sur une infratructure de type Hadoop qui permet d'utiliser une syntaxe semblable au langage SQL. 3, 26, 27, 30, 32, 35, 36, 40–42, 52, 53

N

NameNode Composante principale de Hadoop qui gère l'espace-nom, l'arborescence du système de fichiers et les métadonnées associées. 12, 22

Node Une machine faisant partie d'une grappe (*cluster*) et souvent fabriquée avec du matériel de moindre qualité. ii, iv, xvi, 5–12, 19, 21–26, 30, 31, 34, 35, 42–47, 49–54, 56, 58, 60, 62, 63, 65, 67–71, 73, 74, 76–82, 89, 90, 99, 102

O

OM Ozone manager, composante du système Apache Ozone. 15, 23, 26, 35, 46, 52, 80

Ozone Apache Ozone. i–iv, ix–xii, xv, 2, 3, 13, 14, 16, 18, 19, 21–28, 30–32, 34, 36, 39–41, 46, 47, 53, 55, 58, 60, 61, 63, 65–67, 69, 71, 72, 74–77, 79–82, 89, 99, 100, 102, 104

R

RAFT Replicated and fault tolerant. i–iv, x, xii, xv, xvi, 2–7, 9–11, 13–15, 18, 19, 21, 23–25, 28, 30, 31, 34, 35, 56, 79–82, 99, 104

Ratis Apache Ratis est une librairie Java qui implémente le protocole de consensus RAFT. i, iii, 2, 3, 14, 28, 34, 56, 58, 80, 82

Replica Réplique d'un conteneur (*container*) afin d'en assurer la disponibilité. 16, 18, 19, 21, 81

RPC Remote Procedure Call. Protocol de communication entre les processus de différents serveurs. 7, 9

S

SCM Storage Container Manager, composante du système Apache Ozone. 15, 16, 19, 21, 23–25, 35, 46, 52, 80, 99

Shell Interface qui permet les interactions entre l'utilisateur et un programme ou système d'exploitation grâce à des lignes de commandes. 24, 33, 39, 46, 51

SQL Language de programmation utilisé pour effectuer des requêtes sur des bases de données relationnelles. 26, 30

SQL Server Système de gestion de bases de données relationnelles de la compagnie Microsoft. 29–32, 38, 39, 44

Chapitre 1

PROBLÉMATIQUE

Le présent mémoire s'inscrit alors que nous vivons une ère de changements et bouleversements technologiques, passant ainsi de l'informatique centralisée à l'informatique distribuée, avec son lot d'avantages et d'inconvénients. Le plus grand enjeu des systèmes distribués demeure le problème du consensus qui est directement lié au théorème CAP (*Consistency, Availability and Partition tolerance*) paru en 2000, aussi connu sous le nom de son auteur soit le Théorème de Brewer, stipulant qu'un tel système ne peut garantir que deux contraintes à la fois ; que l'une d'elle doit être sacrifiée au profit des deux autres (Fluri et al., 2018).

Nous pouvons donc distinguer les systèmes distribués en trois (3) classes :

— CA : Cohérence et disponibilité au profit de la tolérance aux partitions comme la plupart des RDBMS tels que Oracle MySQL, Oracle Database et Microsoft SQL Server.
— AP : Disponibilité et tolérance aux partitions au profit de la cohérence comme pour les systèmes Apache Cassandra, Elasticsearch et DynamoDB.
— CP : Cohérence et tolérance aux partitions au profit de la disponibilité comme pour Apache HBase, CockroachDB et etcd.

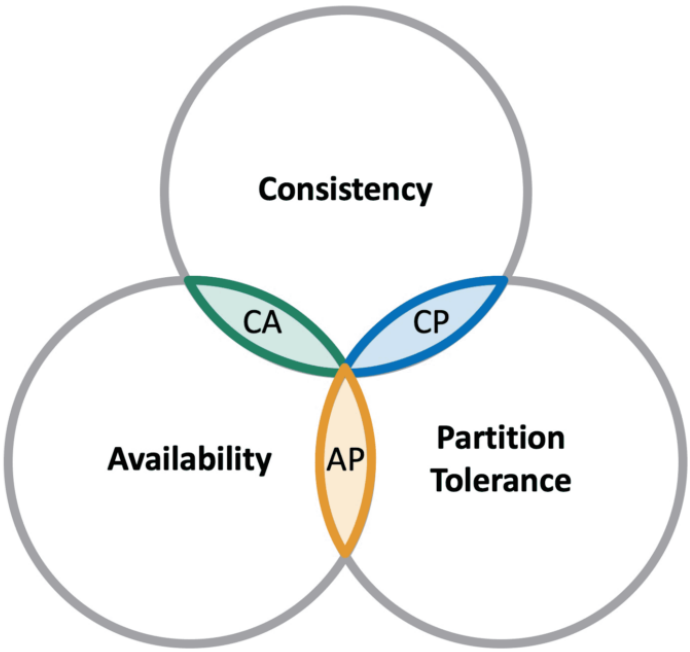

FIGURE 1.1 – Théorème CAP (Brewer) (HAZELCAST, s. d.)

Certains des exemples mentionnés peuvent appartenir à plusieurs classes en fonction de la configuration de ceux-ci.

Avec l'émergence des algorithmes de consensus tel que RAFT, bien des systèmes arrivent maintenant à faire croire qu'un ensemble de machines se comporte comme une seule entité distincte en étant d'accord sur une même valeur (SZTERN, 2015). On peut dire qu'après toutes ces années, certaines règles du théorème CAP ont changé, réalisant ainsi un compromis entre les trois (BREWER, 2012) et c'est ce que nous tenterons de prouver en nous appuyant sur des travaux de recherche antérieurs sur RAFT tout en expérimentant l'implantation de ce protocole au système de stockage d'objets (*object store*) Ozone par l'entremise de la librairie Ratis.

Nous espérons que nos travaux et expérimentations, appuyés par des recherches actives et passées dans ce domaine d'étude, permettront de démocratiser ce qu'est

le consensus distribué, apporter notre apport à la communauté *Open Source* et peut-être même enrichir ou orienter les futurs développements d'Ozone et Ratis. Les résultats de notre recherche que nous vous présentons visent à évaluer l'intégration de RAFT dans Ozone via la librairie Ratis.

1.1 Revue de littérature et objet de recherche

Afin de bien cerner la problématique, il est indispensable de bien comprendre les divers aspects historiques et technologiques entourant les systèmes distribués, touchant plus particulièrement Ozone. Nous allons décrire de façon détaillée les divers éléments primordiaux que nous considérons et abordons dans ce projet de recherche dont l'informatique distribuée, RAFT, Hadoop, Ratis, Ozone et Apache Hive (Hive).

1.1.1 Informatique distribuée

L'ère de l'informatique distribuée ne date pas d'hier, cette thématique est à l'étude depuis plusieurs décennies. Née dans les années soixante-dix, les motivations des créateurs étaient d'augmenter la puissance de calcul et de traitement en partageant des tâches entre plusieurs machines de matériel et ressources hétérogènes (Khan, 2015). À l'époque, les processeurs n'étaient pas aussi performants qu'aujourd'hui, bien que leur vitesse augmentait rapidement (Loi de Moore). L'idée d'éviter le gaspillage de précieuses ressources était alors l'une des motivations de ce champ d'études.

L'avènement d'Internet, dans les années quatre-vingt-dix, est venue accélérer le développement de l'informatique distribuée en rendant disponible un vaste réseau d'ordinateurs (personnels, bureaux et serveurs) qui le composent (N. Ahn et al., 2001). Il était maintenant clair qu'une croissance horizontale d'un système deve-

nait plus logique et flexible qu'une croissance verticale (ajout sans cesse de ressources pour un même serveur), bien que plus difficile. L'un des premiers projets populaires est sans contredit l'expérience scientifique de recherche d'intelligence extraterrestre SETI@home de l'Université de Berkeley. Depuis, plusieurs projets de recherche permettent aux particuliers d'offrir leur puissance de calcul personnelle en installant l'application BOINC sur leurs appareils (ordinateurs, téléphones ou tablettes), propulsée par la plateforme de Science United (Université de Berkeley, 2023).

Évidemment, cette effervescence technologique ne venait pas sans aucune embûche. Cela a pavé la voie pour de nombreux champs de recherche en lien avec les nouvelles problématiques à résoudre telles que la concurrence, la cohérence, la disponibilité et la performance. L'objectif final étant d'avoir une couche d'abstraction rendant simple l'utilisation de tels systèmes par les utilisateurs. Une vérité est certaine et inévitable, il y aura des défaillances notamment des bris d'équipements, latence ou partitionnement du réseau et les systèmes devront les affronter.

Aujourd'hui, cette technologie est partie intégrante de nos vies en raison de l'utilisation d'une multitude de services infonuagiques tels que les courriels, le stockage, les plateformes de diffusion en continu, les cryptomonnaies, etc. Nous ne nous en rendons pas compte, mais derrière ces interfaces, des milliers voire des millions d'ordinateurs mettent leurs ressources en commun afin que ces applications soient disponibles en tout temps et que nos données soient fiables et sécurisées.

1.1.2 RAFT

En 2014, RAFT, un nouvel algorithme de consensus, voyait le jour. Il se voulait plus simple et facile à comprendre que son prédécesseur Paxos, paru en 1998 par le chercheur Leslie Lamport, tout en étant hautement performant et tolérant

aux pannes (Howard et al., 2015). Dans les faits, ces derniers ne seraient pas significativement différents l'un de l'autre, mais RAFT aurait l'avantage d'avoir un processus d'élection de meneur (*leader*) simplifié et une meilleure efficience de reprise après une panne (Howard & Mortier, 2020). Le protocole peut tolérer f pannes pour un *cluster* de $2f + 1$ serveurs (Liu, 2018).

Le consensus est un problème fondamental dans les systèmes distribués puisque ce dernier permet, grâce à son algorithme, d'obtenir un *cluster* dont chacune de ses *nodes* s'entendent sur la validation et la réplication des valeurs des données qui les composent (Ongaro & Ousterhout, 2014). Lorsqu'il y a quorum ou majorité sur une décision, celle-ci est finale et sans appel. Cette façon de faire permet donc la réplication des états des machines et de leurs journaux (*logs*) afin que toutes soient identiques (séquence des commandes), même en cas de partitionnement du réseau, c'est ce que nous appelons la *state machine replication* (Hu & Liu, 2020). Ces mécanismes de consensus ont tendance à infliger une surcharge de latence réseau (délai) en raison de toutes les communications qui vont et viennent entre les différents serveurs du *cluster* (Zhang et al., 2017). De son côté, RAFT décompose le problème de consensus en trois sous-problèmes (Ongaro & Ousterhout, 2014) :

— L'élection du *leader*, c'est-à-dire qu'un nouveau *leader* doit être sélectionné si le *leader* actuel n'est pas en mesure d'effectuer son rôle ;
— La réplication du *log*, à savoir que le *leader* doit accepter les requêtes (entrées de *log*) des clients et les répliquer aux *nodes* du *cluster* en forçant ainsi leur synchronisation avec le sien ;
— La sécurité, en d'autres termes, si une *node* applique une entrée à sa propre *state machine*, alors aucune autre *node* ne peut en appliquer une différente pour un même index.

Dans un premier temps, faisons un survol de la terminologie du protocole de

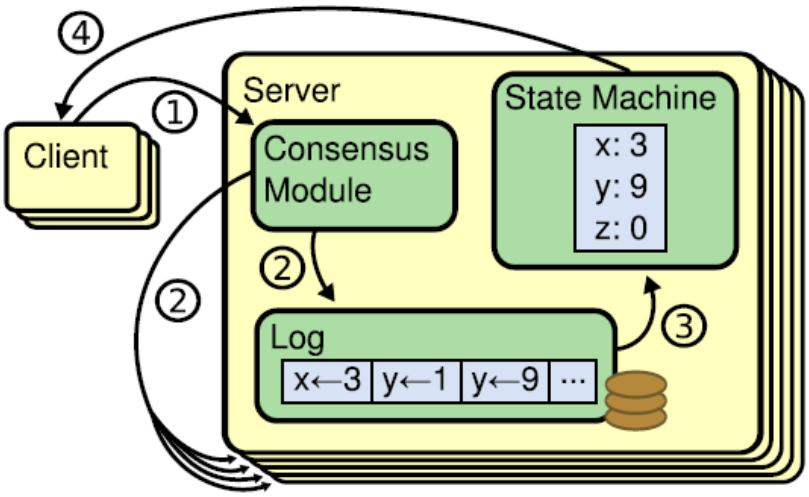

FIGURE 1.2 – Replicated state machine architecture (ONGARO & OUSTERHOUT, 2014).

consensus en question, avant d'expliquer en détails les différents modules de RAFT. La connaissance de ce lexique est importante pour faciliter la compréhension de l'algorithme.

Terme (*term*)

Un terme est un identifiant numérique unique et séquentiel. Chaque terme débute par l'élection d'un *leader* et l'algorithme s'assure de n'avoir qu'un seul *leader* par terme. Le terme est partagé à chacune des communications entre les *nodes*. Si le terme de l'une d'elles est plus petit que celui des autres, elle modifiera son propre terme pour le plus grand. Elle retournera ensuite à l'état de suiveur (*follower*).

Journal (*Log*)

Il est composé d'une liste de commandes (*log entries*). Celui-ci est cohérent tant que toutes les *nodes* s'entendent sur chacune de ces dernières ainsi que de l'ordre, ou séquence, dans laquelle elles apparaissent.

Nodes

Les *nodes* sont les serveurs qui composent le *cluster*, communément appelé le

RAFT *group*. Elles peuvent avoir trois (3) états distincts : candidat (*candidate*), *leader* et *follower*. Le *follower* est une *node* passive qui ne fait que répondre aux commandes RPC (*Remote Procedure Call*). Le *candidate* est une *node* active qui envoie des commandes *RequestVote* lors d'une élection et le *leader*, quant à lui, est une *node* active qui intéragit avec les clients et les *nodes* du *cluster* afin de conserver celles-ci synchronisées en leur envoyant des commandes *AppendEntries* (Howard, 2014).

Battement de coeur (*heartbeat*)

Signal envoyé périodiquement par le *leader* aux *follower*s via une commande *AppendEntries* sans aucune charge afin de maintenir son autorité. (Galery Käser, 2023).

Division du vote (*Split vote*)

Lorsque des *nodes* deviennent à l'état *candidate* en même temps, il y a un risque d'obtenir des résultats égaux. Dans ce cas, il y aura un délai alloué dépassé (*timeout*), le terme sera incrémenté et une nouvelle élection aura lieu.

Commande (*log entry*)

Toute transaction envoyée au *leader* de la part d'un client. Le *leader* notifie alors les *follower*s de la transaction et celle-ci ne sera pas enregistrée (*uncommitted*) tant que la majorité des *nodes* n'auront pas confirmé le traitement de la commande de leur côté. Celle-ci devient alors enregistrée (*commited*) et le *leader* confirme aux *follower*s que la commande a bel et bien été traitée avec succès. Le consensus a été obtenu, c'est ce qui est appelé la réplication du journal (*Log Replication*). Dans sa version la plus simpliste, RAFT utilise seulement deux (2) commandes RPC : *RequestVote* et *AppendEntries*.

Log compaction / compression (*snapshot*)

La commande RPC *InstallSnapshot* permet de compacter le *log* du *leader* et

de l'envoyer via le réseau à un *follower* afin qu'il puisse se mettre à jour, par exemple dans le cas d'une nouvelle *node* ou tout simplement d'une *node* en retard (*lag*). Cette fonctionnalité est nécessaire au protocol afin de réduire le délai de transmission du *log* qui ne fera que grossir au fil du temps (Li et al., 2021).

En deuxième lieu, abordons plus en détails la composante de l'élection du *leader* (*leader election*), c'est-à-dire le processus d'accord entre les membres du *cluster* sur un *leader* potentiel pour un nouveau terme. C'est un impératif, il ne doit y avoir qu'un seul *leader* qui accepte les commandes des clients. Rappelons-nous qu'une *node* peut être dans l'un des trois (3) états suivants : *candidate*, *follower* ou *leader*. Ce dernier est celui qui est responsable des intéractions avec les clients et de la réplication du *log* aux *follower*s qui eux, sont dans un état passif et ne font qu'attendre les requêtes de la part du *leader* ou des *candidate*s (Vanlightly, 2019). Ces derniers passent de l'état de *follower* à *candidate* en détectant l'absence du *leader*, lorsque que leur *election timeout* vient à échéance avant le *heartbeat* du *leader*. Cet état de transition leur permet donc d'effectuer des requêtes auprès des autres *nodes* pour déclencher l'élection. En effet, lorsque les *nodes* ne reçoivent pas de signal du *leader*, celles-ci peuvent devenir *candidate*s au poste de *leader* et demander l'appui (*RequestVote*) des autres *nodes* sous le principe du premier arrivé, premier servi. Le *candidate* devient le *leader* lorsqu'il reçoit une majorité de votes, incluant le sien. Comme le *leader* a beaucoup de responsabilités, le processus de consensus sera plus efficace avec un *leader* dont la machine est plus stable et performante (Lu et al., 2023).

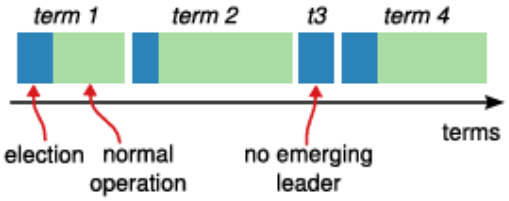

Figure 1.3 – Raft operation terms (Ongaro & Ousterhout, 2014).

Durant l'élection, un *follower* votera uniquement pour un *candidate* dont le terme est plus grand ou lorsque le terme est égal mais dont le journal est plus récent ou identique au sien afin de ne pas perdre de transactions lors d'un changement de *leader*. Dans RAFT, il y a deux paramètres de délais alloués (*timeouts*) qui contrôlent l'élection. Dans un premier temps, il y a celui alloué à l'élection (*election timeout*). Il s'agit du délai maximal attendu par un *follower* avant de devenir *candidate*, qui oscille normalement entre 150 ms et 300 ms, pour ensuite démarrer une nouvelle élection. Le *leader* élu débutera donc l'envoi de messages tels des battements de coeur (*heartbeats*) selon l'intervalle spécifié, soit le délai de battement (*heartbeat timetout*). S'il ne répond plus assez rapidement, le processus d'élection sera enclanché à nouveau.

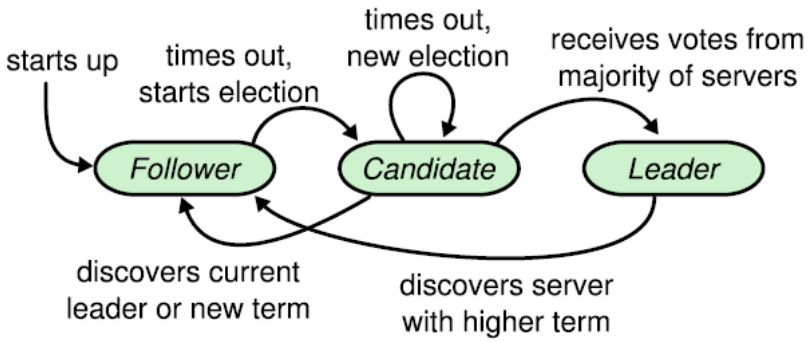

FIGURE 1.4 – Server states (ONGARO & OUSTERHOUT, 2014).

Dans un troisième temps, discutons du processus de mise à jour du *log* de transactions. Une fois élu, le *leader* peut débuter à servir les requêtes provenant des clients. Il ajoute chaque commande à son propre *log* en tant que nouvelle entrée (*uncommitted*) et la propage aux *follower*s à l'aide de commandes RPC de type *AppendEntries* afin que ceux-ci puissent la répliquer à leur tour à leur propre *log* (figure 1.5). Les *follower*s confirment ensuite l'ajout de la commande à leur *log* au *leader* qui pourra l'appliquer officiellement à son tour et modifier l'état de la machine (*committed*). Une entrée peut être appliquée sécuritairement uniquement lorsqu'une majorité de *nodes* l'ont d'abord répliquée. Le résultat peut

ensuite être retourné au requérant.

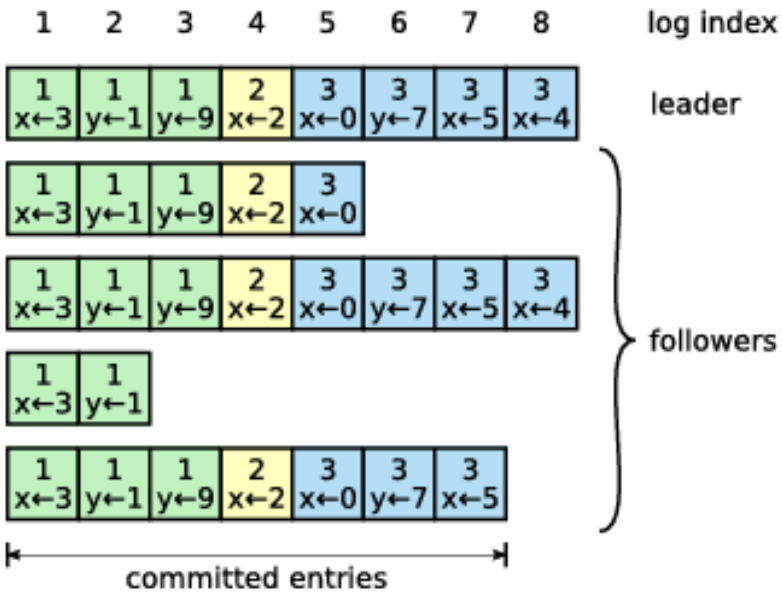

FIGURE 1.5 – Raft log replication(ONGARO & OUSTERHOUT, 2014).

Dans Raft, le *leader* gère les inconsistances des *follower*s en forçant leur synchronisation en répliquant son propre *log* via des commandes *AppendEntries* tant et aussi longtemps que le *log index* n'est pas égal au sien (*log matching*). Ce scénario survient notamment lors de crash, de latence, de partitionnement réseau, de maintenance ou de l'ajout d'une *node* au *cluster*.

Avec toutes ces fonctionnalités et ces cinq (5) règles de sécurité (ONGARO & OUSTERHOUT, 2014), RAFT peut garantir la *state machine* si et seulement si :

- Élection sécuritaire (*election safety*) : Un seul *leader* peut être élu pour un terme donné ;
- Le *leader* ne fait qu'ajouter (*leader append-only*) : Le *leader* peut seulement ajouter des entrées à son journal, il ne peut pas en modifier ni en détruire ;
- Journaux égaux (*log matching*) : Si deux journaux contiennent une même entrée pour un même index et terme, alors nous pouvons conclure que les journaux sont identiques jusqu'à cette même entrée ;

— Complétude du *leader* (*leader completeness*) : Si une entrée au journal est confirmée pour un terme donné, alors celle-ci sera présente dans tous les journaux des futurs *leader*s depuis ce terme ;
— Sécurité de l'état de la machine (*state machine*) : Si un serveur a appliqué une entrée modifiant son état, alors aucun autre serveur ne pourra appliquer une commande différente. Cette sécurité est gérée par les restrictions liées au processus d'élection. Un *candidate* ne peut pas remporter une élection si son journal ne contient pas toutes les entrées confirmées (validation de l'index et du terme de la dernière entrée). Cette vérification se fait *de facto* grâce à l'obligation d'obtenir une majorité de votes de la part des *nodes* du *cluster* pour lesquelles, ces entrées sont aussi présentes dans leurs journaux respectifs. RAFT détermine qu'une entrée, pour deux *nodes* distinctes, sera plus récente selon le terme le plus élevé. Lorsque le terme est identique, le journal le plus long sera considéré comme le plus à jour.

Malgré cela, l'enjeu principal de RAFT et aussi un point de défaillance unique (*single point of failure*), est sa forte dépendance au *leader* et la réplication de son *log*. Ceci est un atout pour sa consistence mais qui peut compromettre sa disponibilité. Par conséquent, il est primordial d'avoir un *leader* fiable pour assurer la stabilité du système (R. Jain et al., 2024). Pour plus d'information à propos des fonctionnalités avancées de RAFT notamment le *joint consensus*, *log repair* et *membership change*, nous vous conseillons fortement la lecture de l'article ou de la thèse de doctorat à ce sujet (Ongaro & Ousterhout, 2014).

1.1.3 Apache Hadoop

Une nouvelle technologie a vu le jour il y a quelques années, sous le terme de lac de données (*data lake*), prônant plutôt l'approche « *schema on read* », ce qui signifie

que la structure est créée et appliquée uniquement lors de la lecture des données (Jankovic et al., 2018). Cette stratégie est opposée au « *schema on write* » des entrepôts de données conventionnels et rigides qui font face à différents enjeux tels que la nature des données et leur stockage, la disponibilité et la performance (Chikh & Warda, 1996). Cette approche se voulait prometteuse en permettant la prise en charge de grands volumes de données structurées et non structurées tout en améliorant l'intelligence d'affaires au sein des entreprises (Llave, 2018).

L'origine de Hadoop (*High-availability distributed object-oriented platform*) provient du projet Apache Lucene, un moteur de recherche libre qui a débuté en 2002 (Chikh & Warda, 1996). Fruit de Doug Cutting qui l'a créé en 2004 en partie grâce aux travaux de recherche publiés par Google et son GFS (*Google File System)*, le projet a littéralement le vent dans les voiles (Shvachko et al., 2010). La tendance vers la démocratisation des données, c'est-à-dire le processus consistant à rendre les données accessibles et faciles à utiliser par un grand nombre d'individus dans une organisation, n'a fait qu'accentuer cette croissance déjà impressionnante (Mone, 2013).

Le système, par son architecture et son système de fichiers (HDFS), permet de gérer une quantité phénoménale de fichiers volumineux en les distribuant en blocs (*blocks*) sur les différentes *nodes* du *cluster*. Le tout est orchestré par la NameNode afin d'en assurer la sécurité et la réplication (Elkawkagy & Elbeh, 2020). L'analyse de toutes ces données de tailles gigantesques se fait par *MapReduce*, un *framework* d'exécution distribuée, qui divise le tout en plusieurs tâches permettant ainsi d'augmenter la puissance de traitement du système (E. P. Jain & Gupta, 2017).

Il est devenu au fil du temps une référence dans l'univers du *Big Data*. Le défi majeur lors de la mise en place d'un *data lake* est de s'assurer qu'il ne devienne pas un marais (*data swamp*). Les enjeux qui le guettent demeurent au niveau de

sa gouvernance afin de garantir la qualité des données, la gestion de la croissance (exponentielle) et le nombre de schémas qui demeurent souvent inutilisés (Ansari et al., 2018). À ce jour, la version la plus récente est la 3.4.1, parue le 18 octobre 2024.

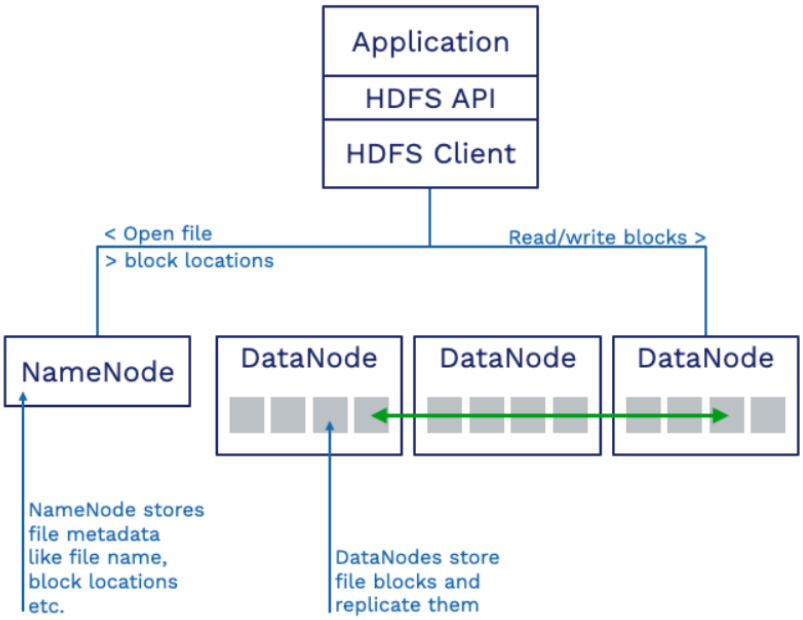

Figure 1.6 – Hadoop architecture (Quobyte, s. d.).

1.1.4 Apache Ratis

Ce projet, incubé et développé encore une fois sous la ASF, a démarré en mars 2016 chez Hortonworks, une compagnie californienne de développement de logiciels basée à Santa Clara. Cette dernière soutenait aussi le projet Hadoop, juste avant de fusionner avec Cloudera en octobre 2018.

Écrit avec le langage Java, l'objectif principal était d'en faire une librairie, et non un service, qui implémente le protocole de consensus RAFT. Il y avait bel et bien le désir de l'utiliser avec Ozone, mais aussi de permettre à n'importe quel système ayant besoin d'un *replicated log* de l'incorporer à son code source. Bien qu'une panoplie d'implémentations du protocole existent déjà dans différents langages

notamment C, C# et Python, ceux-ci ne sont pas faciles à utiliser et à intégrer (Ratis, 2023).

Ratis se distingue par ses quatre (4) plugiciels permettant d'aisément intégrer le transport et la communication (gRPC, Netty et Hadoop RPC), la *state machine*, le RAFT *log* ainsi que plus récemment en 2023, une couche d'observation des métriques (Sze, 2019).

La version 1.0.0 sortie en juillet 2020, a été la première assez stable et mature à être relâchée au grand public. À ce jour, la version la plus récente est la 3.1.2, parue le 13 novembre 2024.

1.1.5 Apache Ozone

La première relâche (*commit* HDFS-7240) d'Ozone a eu lieu en juin 2015, alors qu'il n'était encore qu'une fonctionnalité d'*object store* ajoutée au produit Hadoop (Pareja Prieto, 2022).

Trois ans plus tard, Ozone est devenu officiellement un sous-projet Hadoop pour finalement être séparé complètement de ce dernier en mai 2020 pour le connaître sous la dénomination Ozone. Les principaux objectifs, en plus de la vision *Open Source* de ce projet, étaient d'offrir :

- Un environnement fortement cohérent ;
- Une architecture simplifiée ;
- Une architecture en couches (*layered*) ;
- Un processus de récupération sans tracas ;
- Une interopérabilité avec l'écosystème Hadoop.

Toutes les métadonnées d'Ozone et Ratis sont stockées dans des bases de données RocksDB, une création de Dhruba Borthakur chez Facebook en avril 2012, qui est une bifurcation (*fork*) de LevelDB, un logiciel développé par Google. Ce type

de base de données intégré est reconnu pour ses hautes performances et sa facilité d'intégration (Dong et al., 2017).

Ozone se divise en cinq (5) composantes principales soit OM, SCM, Conteneurs (*Containers*), DN et RECON, tel que vu à la figure 1.7.

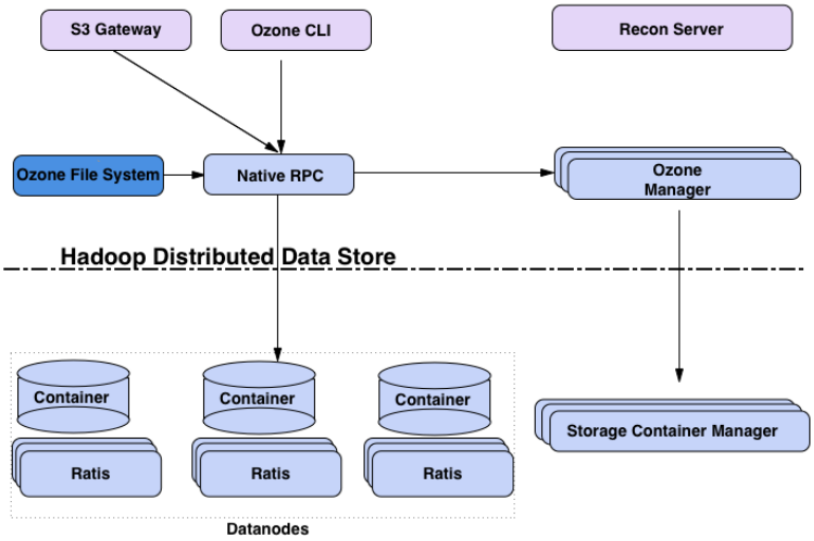

Figure 1.7 – Ozone core components (Ozone, 2023)

Ozone Manager (OM)

Service de gestion de l'espace-nom (*namespace*) qui gère l'organisation des volumes (*volume*s), des sceaux (*buckets*) et des clés (*keys*) (allocation des *block*s). Ce service peut être configuré en haute disponibilité (HA) et synchronisé via RAFT.

Storage Container Manager (SCM)

Service de gestion du cycle de vie des *container*s dont les responsabilités principales sont de gérer le *cluster*, la sécurité, les *block*s et la réplication. Ce service peut être configuré en HA et synchronisé via RAFT.

Ce dernier inclut les services du *Replication Manager* (RM) qui agissent en tâches de fond afin d'assurer la réplication des *container*s, sans toutefois surcharger le *cluster* de ces opérations routinières étant donné que tous ces processus s'exé-

cutent à l'intérieur du SCM.

Il se charge aussi de bien balancer les répliques (*replicas*) vers les noeuds de stockage (*Data Nodes*) selon l'emplacement et l'utilisation de ces dernières ainsi que la priorité de la tâche (*normal* ou *low*).

Les trois (3) types de commandes du RM sont :

— Commandes de réplication de *container* ;
— Commandes de suppression de *replica*s de *container* ;
— Commandes de reconstruction *Erasure Coding* (EC).

Commandes de réplication de *container*

Lorsqu'un *container* est à l'état *UNDER_REPLICATED*, le RM envoie cette commande à une DN qui contient un *replica* du *container* avec l'instruction de se répliquer vers une DN cible qui sera celle ayant le moins de commandes dans sa file d'attente (Cloudera, 2023).

EXTRAIT DE CODE 1.1 – Ozone - Replication Manager - Commande *Replicate*

```
1 # LOG SCM
2 [ReplicationMonitor] INFO LegacyReplicationManager: Container
  ↪ #5043 is under replicated. Expected replica count is 3, but
  ↪ found 2.
3
4 [ReplicationMonitor] INFO LegacyReplicationManager: Sending
  ↪ replicate container command for container #5043 to datanode
  ↪ 5f2009cd-53bf-4d5e-ba6b-b0005082720e{ip: 000.000.000.130,
  ↪ host: servozone01}
```

```
from datanodes [a9522048-a43c-4cdd-97fd-8bb12bb2d816{ip:
↪ 000.000.000.135, host: servozone06},
↪ 4a12af57-593b-4f0e-9a2e-d8dc9a5a5231{ip: 000.000.000.134,
↪ host: servozone05}]

# LOG SERVOZONE01
[ContainerReplicationThread-9] INFO DownloadAndImportReplicator:
↪ Starting replication of container 5043 from
↪ [a9522048-a43c-4cdd-97fd-8bb12bb2d816{ip: 000.000.000.135,
↪ host: servozone06}, 4a12af57-593b-4f0e-9a2e-d8dc9a5a5231{ip:
↪ 000.000.000.134, host: servozone05}]

[grpc-default-executor-458] INFO GrpcReplicationClient: Container
↪ 5043 is downloaded to
↪ /tmp/container-copy/container-5043.tar.gz
[ContainerReplicationThread-9] INFO DownloadAndImportReplicator:
↪ Container 5043 is downloaded with size 1661, starting to
↪ import.
[ContainerReplicationThread-9] INFO DownloadAndImportReplicator:
↪ Container 5043 is replicated successfully
[ContainerReplicationThread-9] INFO ReplicationSupervisor:
↪ Container 5043 is replicated.

# LOG SERVOZONE06
[grpc-default-executor-329] INFO GrpcReplicationService: Streaming
↪ container data (5043) to other datanode
```

```
17 [grpc-default-executor-329] INFO GrpcOutputStream: Sent 1661 bytes
   ↪ for container 5043
```

Commandes de suppression de *container*

Lorsqu'un *container* est à l'état *OVER_ REPLICATED*, le RM envoie cette commande à une DN qui contient un *replica* du *container* avec l'instruction de le détruire. Si la DN est déjà surchargée de tâches, la suppression sera redirigée vers une autre DN ou simplement remise dans la file d'attente pour être relancée ultérieurement (Cloudera, 2023).

EXTRAIT DE CODE 1.2 – Ozone - Replication Manager - Commande *delete*

```
1 # LOG SCM
2 [ReplicationMonitor] INFO
   ↪ hdds.scm.container.replication.LegacyReplicationManager:
   ↪ Container #5035 is over replicated. Expected replica count is
   ↪ 3, but found 4.
3
4 [ReplicationMonitor] INFO
   ↪ hdds.scm.container.replication.LegacyReplicationManager:
   ↪ Sending delete container command for container #5035 to
   ↪ datanode 0d1365d8-5355-494e-a061-b14a51337e32{ip:
   ↪ 000.000.000.133, host: servozone04}
```

Commandes de reconstruction EC

Ces commandes sont les plus coûteuses au niveau des ressources du système lorsque la réplication s'effectue par EC au lieu de RAFT. Les fragments doivent

être reconstruits selon les schémas de parité définis dans la configuration au niveau du *bucket*.

Containers

Le *container* est l'unité de base de Ozone, lequel est géré par le SCM. Il s'agit tout simplement d'une collection de répertoires et de fichiers (*key*s) séparés en plusieurs morceaux (*block*s), comme le démontre la figure 1.8.

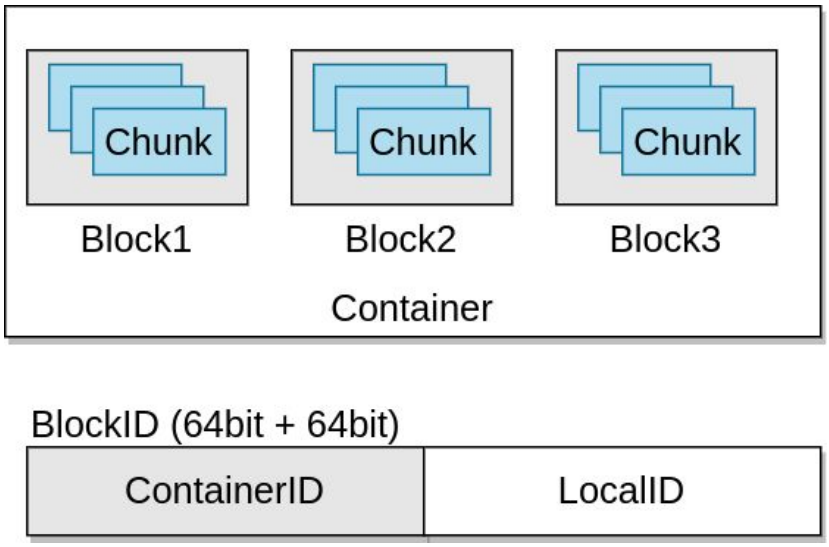

FIGURE 1.8 – Ozone *container* (OZONE, 2023).

Un *container* est soit ouvert (*OPEN*) ou fermé (*CLOSED*). Lorsque le *container* est à l'état ouvert, sa réplication est effectuée par Ratis (3 *replica*s) avec le protocole de consensus RAFT (figure 1.9). Il est *mutable*, c'est-à-dire qu'il est prêt à recevoir des commandes d'écriture (*PutBlock*), de lecture (*ReadBlock*) ou de destruction (*DeleteBlock*) de *block*s par la *node* qui détient le rôle de *leader*. Ce dernier peut emmagasiner, par défaut, jusqu'à 5 Go de données. L'extrait de code 1.3 démontre l'information d'un *container* affichée par la commande *ozone admin container*.

EXTRAIT DE CODE 1.3 – Ozone - *Container* information

```
1 # CMD: ./ozone admin container info 8001
```

```
Container id: 8001
Pipeline id: 52c4909a-87b0-4f28-9df4-5a6266f7b3b4
Container State: OPEN
Datanodes: [41b29633-d924-4b37-bc10-fc9557bd95d8/servozone02,
a9522048-a43c-4cdd-97fd-8bb12bb2d816/servozone06,
b24bcb3c-7b0d-4564-a324-07315c38939d/servozone07]
Replicas: [State: OPEN; ReplicaIndex: 0; Origin:
↪ a9522048-a43c-4cdd-97fd-8bb12bb2d816; Location:
↪ a9522048-a43c-4cdd-97fd-8bb12bb2d816/servozone06,
State: OPEN; ReplicaIndex: 0; Origin:
↪ 41b29633-d924-4b37-bc10-fc9557bd95d8; Location:
↪ 41b29633-d924-4b37-bc10-fc9557bd95d8/servozone02,
State: OPEN; ReplicaIndex: 0; Origin:
↪ b24bcb3c-7b0d-4564-a324-07315c38939d; Location:
↪ b24bcb3c-7b0d-4564-a324-07315c38939d/servozone07]

# CMD: ./ozone admin container list
{
  "state" : "OPEN",
  "replicationConfig" : {
    "replicationFactor" : "THREE",
    "replicationType" : "RATIS"
  },
  "usedBytes" : 6178894,
  "numberOfKeys" : 9,
  "lastUsed" : "2023-08-13T17:10:31.958Z",
  "stateEnterTime" : "2023-08-13T16:15:04.289Z",
  "owner" : "om1",
  "containerID" : 8001,
  "deleteTransactionId" : 0,
  "sequenceId" : 5238,
  "open" : true
}
```

Lorsque le *container* est fermé, il devient *immutable*, c'est-à-dire qu'aucune modification ne peut être effectuée et sa réplication / suppression se fait en mode

asynchrone via le gestionnaire de réplication (RM) avec différents états pendant la transition (*CLOSING, QUASI_CLOSED, DELETING, DELETED, RECOVERING*). Toutes les *nodes* du *cluster* peuvent alors être utilisées pour la lecture.

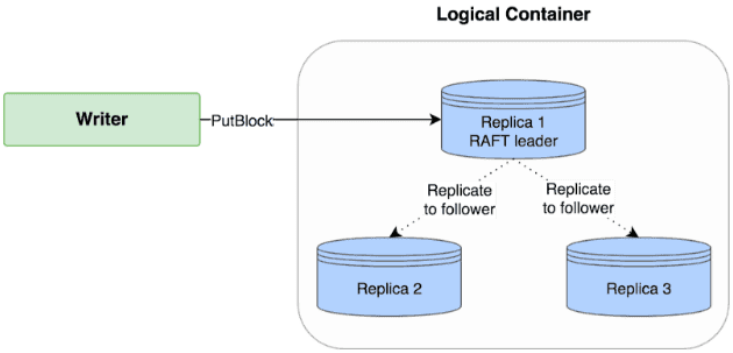

Figure 2: Open Container State Replication Using RAFT

FIGURE 1.9 – Open *container* state replication using RAFT (AGARWAL, 2018)

Les *container*s peuvent avoir différents états de santé (8) pendant leur cycle de vie :

- *UNDER_REPLICATED* : nombre insuffisant de *replica*s ;
- *MIS_REPLICATED* : nombre insuffisant de baies (*racks*) ;
- *OVER_REPLICATED* : plus de *replica*s que requis ;
- *MISSING* : absent, sans aucun *replica* ;
- *UNHEALTHY* : fermé ou quasi-fermé avec des *replica*s comportant différents états ;
- *EMPTY* : sans aucun *block* ;
- *OPEN_UNHEALTHY* : ouvert avec des *replica*s comportant différents états ;
- *QUASI_CLOSED_STUCK* : preque fermé mais avec un nombre insuffisant de DN.

C'est grâce aux rapports de *container*s des *nodes* au SCM qu'Ozone offre un avantage sur Hadoop et sa problématique de gestion des petits fichiers et des mil-

liers de rapports envoyés des *nodes* vers la NameNode, provocant une utilisation et un besoin excessif de mémoire (Aggarwal et al., 2022). Cette façon de faire permet à Ozone d'atteindre le milliard de *key*s, ce qui est bien au-delà de Hadoop, qui recommande un maximum de trois cents (300) millions de fichiers pour une NameNode (Vadivelu et al., 2020).

*Key*s et *Block*s

Les répertoires et les fichiers se nomment des *key*s dans le jargon Ozone. Ces derniers sont divisés en plusieurs *block*s. Le nombre de *block*s est calculé en fonction de la taille de la *key* et de la taille maximale d'un *block*. Celle-ci est déterminée selon la configuration d'Ozone et par défaut elle est de 64 Mo. L'extrait de code 1.4 démontre une *key* de type répertoire tandis que l'extrait de code 1.5 une *key* de type fichier.

EXTRAIT DE CODE 1.4 – Ozone - *Key* (répertoire)

```
{
  "volumeName" : "volo3fs",
  "bucketName" : "buckhive",
  "name" : "external/colbec01/",
  "dataSize" : 0,
  "creationTime" : "2023-06-16T12:20:04.718Z",
  "modificationTime" : "2023-06-16T12:20:04.718Z",
  "replicationConfig" : {
    "replicationFactor" : "THREE",
    "requiredNodes" : 3,
    "replicationType" : "RATIS"
  },
  "metadata" : { }
}
```

EXTRAIT DE CODE 1.5 – Ozone - *Key* (fichier)

```
{
```

```
2   "volumeName" : "volo3fs",
3   "bucketName" : "buckhive",
4   "name" : "external/colbec01/colbec01.csv",
5   "dataSize" : 90939455,
6   "creationTime" : "2023-06-16T12:20:08.867Z",
7   "modificationTime" : "2023-06-16T12:20:16.349Z",
8   "replicationConfig" : {
9     "replicationFactor" : "THREE",
10    "requiredNodes" : 3,
11    "replicationType" : "RATIS"
12  },
13  "metadata" : { }
14 }
```

Lors de la création d'une *key*, OM demande l'allocation d'un ou plusieurs nouveaux *block*s au SCM. Ce dernier valide dans quel *container* il est préférable de les stocker puis il génère un identifiant de *block* composé du *ContainerID* ainsi que d'un identifiant local pour chacun d'eux (figure 1.8). Le client se connecte ensuite à la DN (*leader*) qui détient le *container* via un tuyau (*pipeline*) et gère sa copie locale pour ensuite la répliquer aux autres DNs (Ozone, 2023).

C'est ce qui fait la force d'Ozone car même si une DN détient des millions de *block*s, ce sont uniquement les statuts des *container*s qui seront envoyés au SCM et non ceux des *block*s.

*Pipeline*s

Les *pipeline*s sont des regroupements de *nodes* selon un facteur de réplication (1 ou 3). Ce dernier peut être modifié dans la configuration d'Ozone ou nous pouvons le spécifier directement lors de l'ingestion d'un fichier. Les *pipeline*s sont gérés par le SCM et la réplication se fait en utilisant le protocole de consensus RAFT ou par la méthode EC, mais nous privilégions la première dans le cadre de ce mémoire. Le *pipeline* devient alors un RAFT *Group* pour lequel il y a un *leader*

et deux *follower*s avec un quorum de deux (2) pour la majorité et le processus de réplication. Dans l'extrait de code B.2, nous pouvons voir le résultat de la commande shell *ozone admin datanode list* qui permet de visualiser les *nodes* du *cluster*, leurs états (opérationnel et santé) ainsi que les *pipeline*s auxquels elles sont associées et avec quels rôles.

Le *pipeline* peut avoir quatre (4) états : *ALLOCATED*, *OPEN*, *CLOSE* OU *DORMANT*, comme le montre la figure 1.10. L'extrait de code B.1 en Annexe C, quant à lui démontre des exemples de création, finalisation et fermeture de *pipeline*s provenant du journal du SCM.

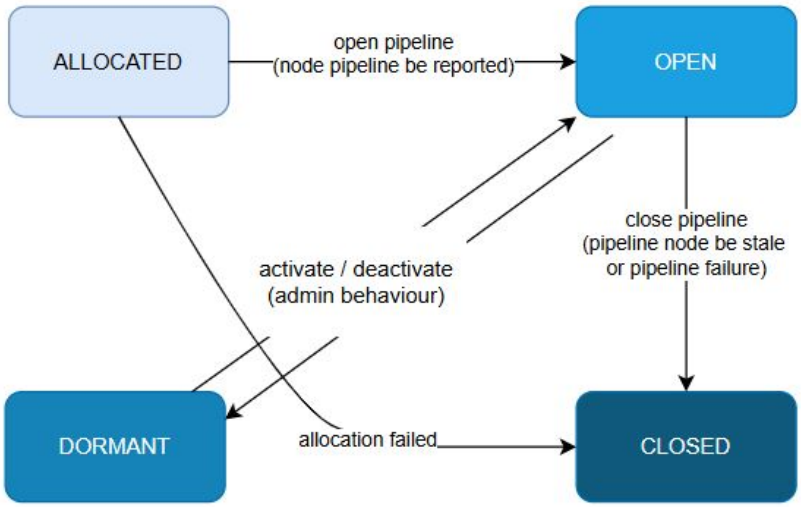

FIGURE 1.10 – Ozone - Cycle de vie d'un *pipeline* (OZONE, 2023)

Pour une performance accrue d'écriture, un mode appelé *Multi-RAFT* est possible en permettant aux DNs de faire partie de plusieurs *pipeline*s à la fois afin d'augmenter la disponibilité des *container*s et des *pipeline*s, améliorer la bande passante engendrée par l'IO et optimiser l'utilisation des *nodes* (CHENG, 2020).

Nous utilisons la valeur par défaut qui limite à deux (2) *pipeline*s par *node* afin de ne pas surcharger inutilement les impacts de nos sabotages. Notre *cluster* peut donc utiliser quatre (4) *pipeline*s en même temps au lieu d'un seul, voir l'équation

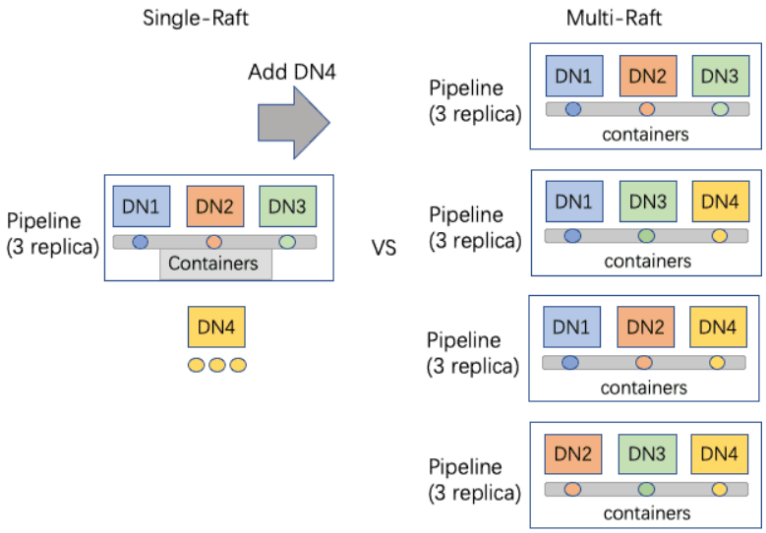

FIGURE 1.11 – Ozone - Multi-RAFT (CHENG, 2020)

1.1 pour le calcul.

$$\text{Pipelines} = \left(\frac{\text{Nombre de nodes du cluster} \times \text{Limite}}{3} \right) \tag{1.1}$$

DNs

Les DNs sont des *nodes* sur lesquelles un service *DataNode* s'exécute. Elles sont au coeur de la disponibilité et la robustesse du système. Toutes les données (*key*s) d'Ozone sont dispersées en *block*s dans les *container*s qui eux sont stockés et répliqués sur les *nodes*.

Elles sont en charge d'effectuer les rapports de leurs *container*s au SCM afin d'en assurer la réplication par RAFT ou le RM selon l'état de chaque *container*.

Recon

Recon est une composante optionnelle d'Ozone mais tellement utile. Il s'agit d'une interface web et d'un service d'analyse qui permet une meilleure visibilité du contenu du *cluster*, tel que vu à la figure 1.12.

Recon collecte toutes les données du *cluster*, en mode asynchrone pour la récep-

tion de *snapshots* de OM et en mode synchrone pour les *heartbeats* des *nodes*. Il permet la consultation de divers tableaux de bords notamment un aperçu, les *nodes* et les *pipelines* afin de mieux surveiller l'état global du système.

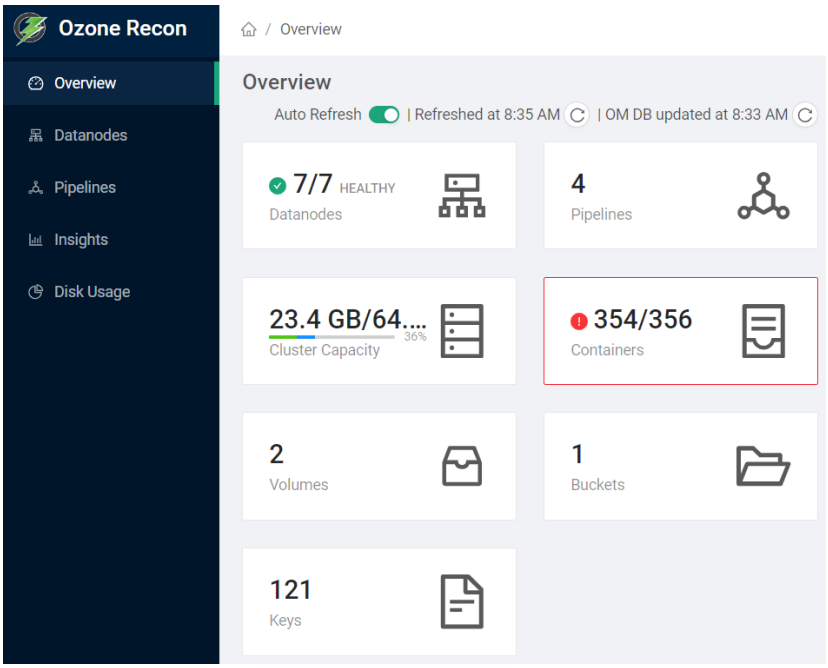

FIGURE 1.12 – Ozone Recon (Aperçu)

À ce jour, la plus récente version de Ozone est la 1.4.1, parue le 24 novembre 2024.

1.1.6 Apache Hive

Hive est un projet développé depuis octobre 2010, originalement par la compagnie Facebook maintenant connue sous le nom Meta, afin de pouvoir exécuter des requêtes de type *Structured Query Language* (SQL), ou plutôt HiveQL, sur un lac de données (*data lake*) tel Hadoop, comme s'il s'agissait d'un véritable entrepôt de données traditionnel (CAMACHO-RODRÍGUEZ et al., 2019). Les systèmes de fichiers Ofs et O3fs comptabiles avec Hadoop pouvant être utilisés sans effort pour Ozone, il est donc tout naturel de l'utiliser comme client pour pour tester Ozone (OZONE, 2023).

Lorsqu'une commande HiveQL est lancée, Hive sert d'interface entre l'utilisateur et Hadoop afin de convertir, ou plutôt compiler, cette dernière sous forme de programmes *MapReduce*, le langage reconnu par le système distribué Hadoop afin de traiter l'information disponible. Le logiciel supporte pratiquement tous les types de données, primitifs ou complexes, notamment *integer*, *float*, *maps* et *structs* (Thusoo et al., 2010). Le dépôt central des métadonnées de Hive se trouve dans ce que l'on appelle le *Metastore*, qui lui permet de faciliter l'écriture des commandes sans avoir à fournir toute la définition des données à récupérer. Par défaut, le type de base de données relationnelle utilisé est celui d'un autre projet ASF, soit Apache Derby. Celle-ci n'est pas recommandée dans un environnement de production car elle permet seulement une session active de Hive.

À ce jour, la plus récente version de Hive est la 4.0.1, parue le 2 octobre 2024.

1.2 Question de recherche et justification

Le principal objectif de ce mémoire consiste à évaluer la fiabilité et la cohérence du système distribué Ozone tout en assurant une haute disponibilité.

Cet objectif principal peut être décliné en trois objectifs secondaires :

- Valider si Ozone comporte des déficiences lorsqu'il est utilisé en mode opérationnel au quotidien ;
- Vérifier si Ozone, soumis à de fortes pressions qui reflètent des situations et défaillances hors du commun (sabotages), peut continuer à fonctionner normalement ;
- Déterminer les défaillances qui ont le plus d'impact sur le fonctionnement d'Ozone en utilisant les tests statistiques de Shapiro-Wilk et Mann-Whitney-Wilcoxon.

Les expérimentations réalisées par le présent mémoire ont été soigneusement sé-

lectionnées dans le but de répondre à cette question : Est-ce qu'Ozone, avec son intégration du protocole de consensus RAFT via la librairie Ratis, nous permet d'obtenir un système distribué fiable et cohérent ?

Chapitre 2

MÉTHODOLOGIE

Ce deuxième chapitre présente la méthodologie utilisée dans le cadre de ce mémoire. Dans un premier temps, nous aborderons la démarche que nous avons mise en place afin d'effectuer nos expérimentations. Ensuite, nous vous présenterons le matériel et les logiciels requis pour générer les données qui, après analyse, nous permettront d'émettre nos conclusions face aux objectifs que nous avons fixés préalablement.

2.1 Démarche

Tout d'abord, nous utiliserons cinq (5) tables provenant de cinq (5) systèmes d'information de l'École nationale de police du Québec, dont les données personnelles nominatives utilisées dans le cadre de ce projet n'ont pas été conservées ou ont été anonymisées par une fonction de hachage cryptographique. Ces tables, contenant approximativement entre 50 000 et 1 200 000 enregistrements chacune, ont été mises en commun sur un même serveur Microsoft SQL Server (*SQL Server*) de développement, tel un entrepôt de données traditionnel dont le schéma rigide a été prévu afin d'y accueillir des données préalablement formatées (Jankovic et al.,

2018). Ces données seront extraites de leurs tables vers des fichiers distincts pour être ingérés dans Ozone. Nous assurerons l'intégrité de départ en effectuant un contrôle de redondance cyclique (CRC), méthode largement utilisée dans Hadoop (Xia, Yun-Hao et al., 2017), sur les différents fichiers générés. Nous pourrons ensuite appliquer un schéma à la demande sur les fichiers d'Ozone en utilisant une panoplie de structures de données mises à notre disposition grâce au client Hive (Jankovic et al., 2018), un logiciel permettant d'utiliser le langage HQL dans un environnement distribué afin d'y effectuer diverses opérations. Nos environnements *SQL Server* et Ozone contiendront alors exactement les mêmes tables avec les mêmes données.

Dans un deuxième temps, nous allons élaborer plusieurs requêtes SQL, tel un scénario, comprenant diverses opérations sur les données notamment des insertions (*insert*), des mises à jour (*update*) et des suppressions d'enregistrements (*delete*) dans les tables. Ce scénario sera converti en procédure stockée sur le serveur *SQL Server* puis exécuté. Les tables seront à nouveaux extraites afin de les conserver à titre de référence. Elles serviront de comparaison avec celles qui seront extraites de Hive, encore une fois avec l'utilisation du CRC.

En troisième lieu, nous allons élaborer un script Hive comprenant les mêmes opérations que celles appliquées sur les tables de la base de données *SQL Server*, afin d'y générer le même ensemble de données. Celles-ci permettront de bien garnir le journal (*log*) de RAFT, c'est-à-dire la réception des commandes du client par le *leader* et la retransmission aux *follower*s (Woos et al., 2016). Si ce n'était que cela, ce serait un peu trop simpliste. Tout cela va se corser car nous allons provoquer, ou plutôt saboter, intentionnellement divers types de pannes au niveau des *nodes* de notre infrastructure Ozone grâce à différents outils technologiques à notre disposition (crash, latence, partition réseau, etc.). Ces sabotages, dont la séquence peut être consultée au tableau 2.1, nous permettront de tester les divers

événements normalement gérés par le protocole de consensus RAFT permettant de conserver la cohérence du système tant que le quorum, tel que vu à l'équation 2.1, demeure intact, ce qui est un bon défi surtout en minimisant la dégradation de la performance (J.-S. Ahn et al., 2019). En effet, RAFT sacrifiera la disponibilité (temps de réponse) au profit de la cohérence et la tolérance au partitionnement du réseau.

$$\text{Quorum} = \left(\frac{\text{Nombre de nodes}}{2}\right) + 1 \tag{2.1}$$

Tableau 2.1 – Séquence des sabotages

Sabotages		
Séquence	Description	Métriques (Variables)
1	Surutilisation du processeur	Pourcentage et durée
2	Surutilisation de la mémoire	Nombre de Go et durée
3	Saturation de l'espace disque	Taille du fichier (Go)
4	Arrêt d'une *node*	-
5	Délai réseau	Délai (ms) et durée
6	Perte de paquets	Pourcentage et probabilité
7	Réduction de bande passante	-
8	Redémarrage d'une *node* avec délai	Délai (s)
9	Corruption de paquets	Pourcentage
10	Redémarrage immédiat d'une *node*	-

Enfin, nous procéderons à la validation des données générées pendant nos simulations avec Ozone. En effet, si tout se passe exactement comme la théorie le stipule, nous devrions obtenir des tables identiques correspondant aussi en tous points à celles provenant du serveur *SQL Server*. Nous ferons alors l'extraction des données dans des fichiers. Pour valider l'exactitude de ceux-ci, nous effectuerons encore une fois un CRC sur chacun d'eux en les comparant à ceux provenant de *SQL Server* suite à l'exécution de la procédure stockée.

En résumé, nos moyens pour arriver à nos fins sont :

— Mettre en place un *cluster* de serveurs virtualisés ;

— Installer et configurer un environnement Ozone en mode distribué ;

— Installer et configurer le client Hive ;

— Sélectionner les sources de données ;

— Schématiser les liens entre les diverses tables ;

— Extraire les données initiales et suite aux opérations (procédure stockée) depuis Microsoft SQL Server ;

— Ingérer les données dans Ozone et valider les CRC de départ ;

— Effectuer cent (100) itérations du script d'opérations CRUD lancé par Hive dont la moitié (50) comportent dix (10) sabotages distincts. Après chacunes d'elles, exporter les données.

— Comparer les CRC des fichiers Ozone et *SQL Server*.

2.2 Matériel et logiciels

Lors de la réalisation du projet, nous avons utilisé du matériel et différents logiciels. Les serveurs ont été virtualisés à partir d'un environnement VMware vSphere (figure 2.1) de huit (8) hôtes physiques branchés sur une architecture de stockage réseau (SAN).

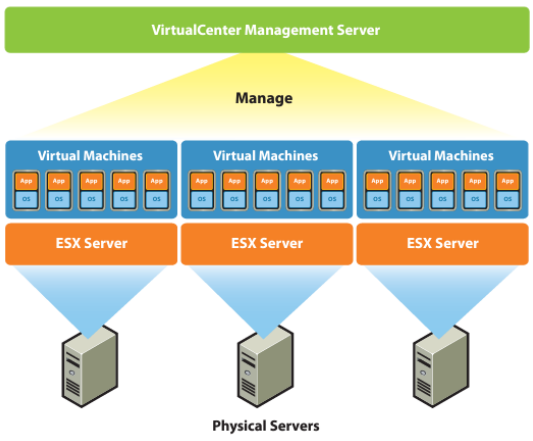

FIGURE 2.1 – VMware vSphere (VMWARE, 2024)

Le tableau 2.2 présente la liste exhaustive des logiciels et petits utilitaires utilisés tout au long de la concrétisation du projet mais aussi pour la rédaction de ce mémoire. Certains de ceux-ci, bien qu'ils puissent paraître futiles à première vue, ont grandement contribué à la collecte de données et aux tests des sabotages afin de nous assurer de leur bon fonctionnement.

Tableau 2.2 – Liste des logiciels utilisés

Nom	Version	Description
Apache Ozone	1.3.0	Version stable, relâche 18 décembre 2022
Apache Hive	2.3.4	Version stable, relâche 7 novembre 2018
Oracle MySQL	5.7.26	Version stable du dépôt Ubuntu.
Java (OpenJDK)	1.8.0	Version Open source de Oracle JDK 8.
Microsoft SQL Server	2016 (SP2-CU8)	Version à jour pour les sources de données.
Putty	0.78	Version stable de 29 octobre 2022.
Remote Desktop Manager	2024.1.25	Contrôle à distance. Utilisé à partir d'avril 2022.
Notepad++	8.4.8	Version stable mise à jour le 24 décembre 2022.
collectl	4.3.1-1	Service de collection de métriques systèmes.
PingInfoView	2.30	Ping de plusieurs machines à la fois.
iperf	2.0.13	Outil pour la mesure de la bande passante maximale entre deux machines sur le réseau.
pdsh	2.31	Outil pour distribuer des commandes shell sur plusieurs hôtes à la fois.
RHash	1.3.9	Calcul et vérification des *hash* notamment CRC32, SHA512 et MD5.
RStudio	2023.9.1	Logiciel libre d'analyses statistiques.

Chapitre 3

EXPÉRIMENTATION

Dans le cadre de notre emploi à l'École nationale de police du Québec, un *cluster* Ozone comportant plusieurs serveurs virtualisés (VMWare vSphere) sous la distribution Ubuntu du système d'exploitation Linux est disponible afin que nous puissions y effectuer nos travaux de recherche. Le nombre de serveurs, soit sept (7), permet tester la tolérance aux pannes de RAFT. En effet, selon la théorie que stipule l'algorithme (Ongaro & Ousterhout, 2014), l'environnement peut demeurer cohérent tant que nous arrivons à conserver un quorum au niveau des différents *pipeline*s répliqués par Ratis (RAFT *group* de trois (3) *nodes*). De notre côté, la configuration Multi-RAFT d'Ozone permet jusqu'à quatre (4) *pipeline*s simultanément.

Notre environnement de simulation permet donc de perdre momentanément plusieurs serveurs sans que les données ne soient affectées. Les spécifications des serveurs sont minimales car nous ne recherchons pas la performance mais la cohérence.

3.1 Infrastructure

Nos serveurs sont tous identiques excepté SERVOZONE01. Le tableau 3.1 démontre effectivement que ce dernier contient le double de mémoire et de processeurs. La raison est bien simple, c'est que cette machine héberge aussi les services OM, SCM, Recon, DataNode en plus de Hive et son *metastore*. C'est aussi à partir de ce serveur que le client Hive (CLI) est utilisé pour l'exécution des scripts.

Tableau 3.1 – Infrastructure - Liste des serveurs

Serveur	**Ressources**	**Services**
SERVOZONE01	2 vCPU, 8 Go RAM, 2 HDD (25 Go et 10 Go)	SCM, OM, RECON et DN
SERVOZONE02	1 vCPU, 4 Go RAM, 2 HDD (25 Go et 10 Go)	DN
SERVOZONE03	1 vCPU, 4 Go RAM, 2 HDD (25 Go et 10 Go)	DN
SERVOZONE04	1 vCPU, 4 Go RAM, 2 HDD (25 Go et 10 Go)	DN
SERVOZONE05	1 vCPU, 4 Go RAM, 2 HDD (25 Go et 10 Go)	DN
SERVOZONE06	1 vCPU, 4 Go RAM, 2 HDD (25 Go et 10 Go)	DN
SERVOZONE07	1 vCPU, 4 Go RAM, 2 HDD (25 Go et 10 Go)	DN

3.2 Configurations

Pour les besoins du projet et en fonction des ressources disponibles pour chacune des *nodes* du *cluster*, nous avons modifié certains paramètres dans le fichier de configuration ozone-site.xml (tableau 3.2). L'objectif est d'engendrer plus rapidement les changements au niveau de RAFT et de voir sans tarder les effets des sabotages sur le *cluster*.

Tableau 3.2 – Configurations - Ozone - Paramètres modifiés

Propriété	**Défaut**	**Valeur**
ozone.scm.dead.node.interval	10m	3m
ozone.scm.stale.node.interval	5m	1m
ozone.scm.container.size	5GB	1GB
ozone.scm.pipeline.creation.auto.factor.one	true	false
hdds.datanode.dir.du.reserved.percent	0	0.05
ozone.recon.om.snapshot.task.interval.delay	10m	5m

Du côté de Hive, nous avons aussi modifié certains paramètres au niveau du fichier de configuration hive-site.xml (tableau 3.3). De plus, nous utilisons plutôt MySQL, le moteur de base de données open-source acquis par Oracle lors du rachat de Sun Microsystems en 2010, pour le *Metastore*.

Tableau 3.3 – Hive - Paramètres modifiés

Propriété	Défaut	Valeur
hive.exec.mode.local.auto	false	true

3.3 Données

Plusieurs systèmes d'information sont actuellement disponibles à l'École nationale de police du Québec et renferment une grande quantité de données, parfois homogènes et d'autres fois hétérogènes, qui ne sont pas exploitées de la façon la plus optimale. Dans le cadre de ce projet, nous en avons sélectionnés quelques-uns qui sont intéressants pour de futures recherches du Centre de recherche et de développement stratégique (CRDS) de l'École. Cela nous permet de mieux cerner la complexité que requiert la mise en place d'un *data lake* et l'utilisation de la méthode *schema on read*.

3.3.1 Sélection des sources

Il est important de mentionner que l'utilisation des sources de données a été octroyée par le Responsable des technologies et de la sécurité de l'information (STSI) de l'École nationale de police du Québec.

COBA Collégial – Système de gestion des dossiers scolaires

Nouvellement implanté à l'École en remplacement d'un vieux système nommé eduZone, ce dernier permet de gérer toutes les facettes reliées aux activités pédagogiques de niveau collégial notamment l'admission, la planification des cours et le suivi des résultats scolaires.

Détenteur de l'actif informationnel : Registraire

Moodle – Plateforme d'apprentissage en ligne (LMS)

Logiciel libre modulaire permettant l'élaboration et la diffusion de contenus pédagogiques en ligne par la biais d'interactions entre les pédagogues, les apprenants et les nombreuses ressources mises à la disposition de ces derniers.

Détenteur de l'actif informationnel : Responsable des activités pédagogiques

CECAP – Logiciel de gestion de la relation avec la clientèle (CRM)

Cette application a été développée à l'École nationale de police du Québec il y a une quinzaine d'années et son nom représente le Centre d'évaluation des compétences et aptitudes professionnelles. Il s'agit d'un département dont la mission est d'offrir leurs services aux organisations de la sécurité publique (policiers, pompiers, etc.), que ce soit en contexte de recrutement, de promotion ou de développement (tests de connaissances, tests psychométriques, entrevues, etc.).

Détenteur de l'actif informationnel : Responsable des activités de formation en sécurité publique

3.3.2 Schématisation

Afin de protéger les données personnelles que contiennent ces systèmes, les données nominatives n'ont pas été conservées ou ont été anonymisées par la fonction de hachage cryptographique *Secure Hash Algorithm* (SHA-2) publiée par le *National Institute of Standard and Technology* (NIST) et utilisée par le gouvernement des États-Unis comme un standard fédéral de traitement de l'information (Pham et al., 2022). Nous l'utilisons pour la préparation des données à extraire de *SQL Server* via la fonction *HASHBYTES*. De plus, nous épurons et réduisons volontairement les ensembles de données (tables ou vues) afin de ne conserver que l'information nécessaire pour l'analyse des différentes dimensions jugées intéressantes et ainsi en produire des résultats fictifs.

Nous avons cinq (5) tables :

— COBA.COLELE01 : Table des étudiants ;

— COBA.COLBEC01 : Table des notes des étudiants ;

— MOODLE.MDL_USER : Tables des étudiants ;

— MOODLE.MDL_GRADE_GRADES : Table des notes des étudiants ;

— CECAP.VENPQ_MPULSE_RAPPORT : Table des résultats agrégés du test psychométrique M-Pulse.

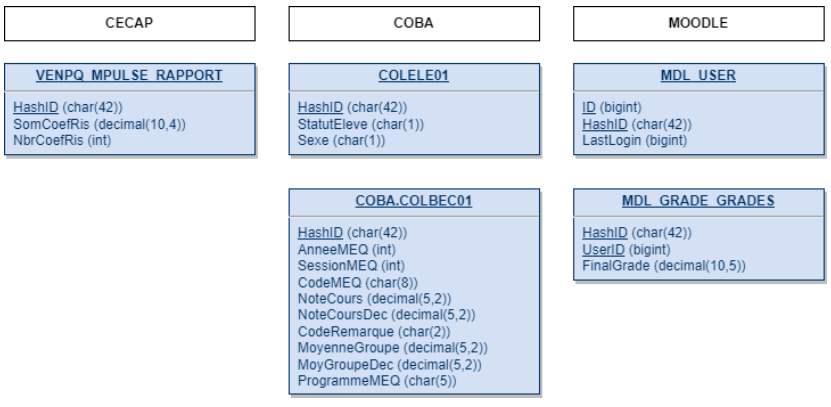

Figure 3.1 – Schéma des tables des divers systèmes

3.3.3 Export des données

Les données sont exportées du serveur *SQL Server* à l'aide de requêtes et nous enregistrons les résultats dans plusieurs fichiers au format CSV (séparateur point-virgule). Nous obtenons ainsi cinq (5) fichiers totalisant 2 127 329 enregistrements et d'une taille combinée de 344.85 Mo, comme le présente le tableau 3.4.

Tableau 3.4 – Renseignements sur les tables

Table	Nombre d'enregistrements	Taille (Ko)
COLELE01	55 348	7 351
COLBEC01	536 203	88 809
MDL_USER	55 561	7 788
MDL_GRADE_GRADES	1 221 457	179 142
VENPQ_MPULSE_RAPPORT	258760	70 040

Nous téléversons ensuite les fichiers vers le serveur SERVOZONE01 avec l'utilitaire PSCP de Putty. Nous conservons ces fichiers dans un répertoire distinct car ils seront utiles pour la validation des CRC suite aux éxécutions en mode Opérations ou Sabotages.

3.3.4 Ingestion des données

L'ingestion de données dans Ozone se fait via le shell du système de fichier *fs* et la commande *put*. Par exemple, la commande suivante importe le fichier test.csv dans le répertoire nommé test de notre *bucket* appelé buckhive appartenant au *volume* volo3fs :

ozone fs -put test.csv o3fs ://buckhive.volo3fs.localhost/test

Nous procédons de cette façon pour l'ingestion de toutes nos tables *SQL Server*. Il est important que nos fichiers soient importés dans le chemin d'accès *o3fs ://buckhive.volo3fs.localhost/* afin que ces derniers soient dans le bon système de fichiers, tel que spécifié lors de la configuration d'Ozone. Le système de fichiers o3fs a été choisi au lieu de ofs, mais la seule différence est que ce dernier peut être utilisé

partout dans Ozone tandis qu'avec o3fs, ce système de fichiers est uniquement réservé à l'emplacement spécifié dans le fichier de configuration. Par la suite, il nous est possible de créer des répertoires distincts pour chacune des tables. L'utilitaire supporte la plupart des commandes Linux usuelles notamment *ls, mkdir et rm.*

3.3.5 Création des tables externes

Une fois les fichiers ingérés dans Ozone, nous pouvons procéder à la création des tables logiques à partir de Hive et du script HQL. Cela permet d'appliquer un schéma sur un fichier de données que nous désirons utiliser comme s'il s'agissait d'une table (*schema on read*).

Nous créons deux bases de données distinctes dans Hive : o3fs et ozone. La première contient nos cinq (5) tables de base (COLELE01, COLBEC01, MDL_USER, MDL_GRADE_GRADES et VENPQ_MPULSE_Rapport) et la seconde contient les tables temporaires utilisées pendant le scénario avec ou sans sabotage. Ces dernières sont créées dynamiquement en tant que tables gérées (*managed tables*) transactionnelles aux propriétés ACID afin de pouvoir y effectuer des opérations CRUD (Cloudera, 2021). L'exemple présenté à la figure 3.2 démontre bien que l'emplacement de la table COLELE01 est dans Ozone (*location*).

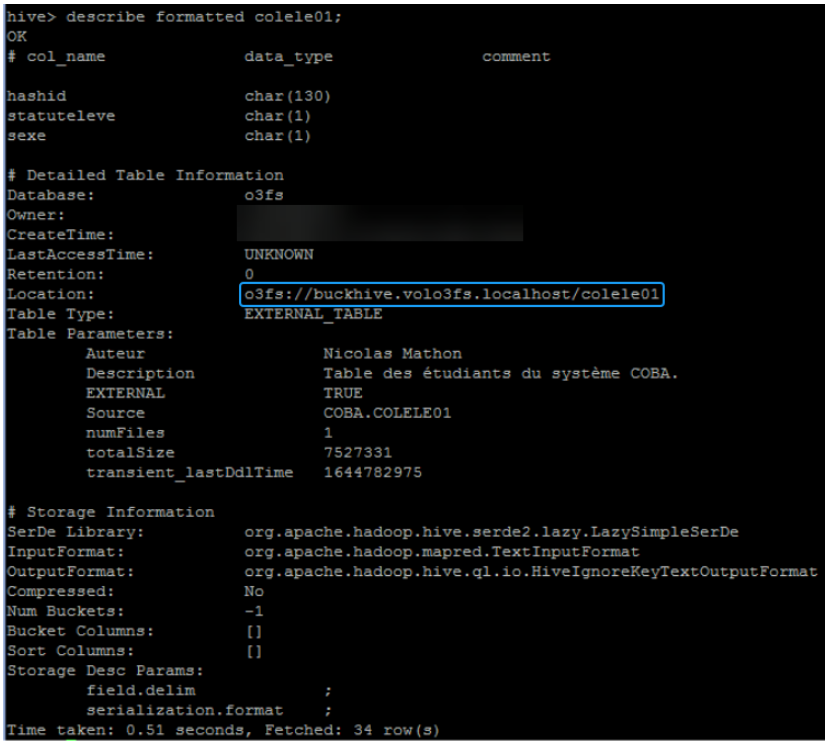

FIGURE 3.2 – Détails d'une table externe

3.4 Scénario

Le scénario représente l'ensemble des opérations CRUD effectuées sur les données d'Ozone. Le scénario est identique peu importe le mode de lancement du script (Opérations ou Sabotages). La seule différence est l'activation des sabotages à certains moments clés lors du mode Sabotages. Le scénario est divisé en dix (10) parties (P) afin de pouvoir mesurer et comparer les effets des sabotages. Toutes les opérations se passent au niveau de la base de données Hive nommée ozone.

1. Suppression des tables temporaires (si elles existent).

   Ces tables sont utilisées plus loin dans le script HQL car certaines opérations requièrent l'utilisation d'une table intermédiaire.

2. Création des tables gérées du schéma.

   À chaque exécution du script crud.sh, nous recréons les tables à partir des

fichiers ingérés préalablement (base de données Hive o3fs) afin de ne pas avoir à refaire l'ingestion à chaque fois.

3. **Partie 1**

   *** SABOTAGE #1 - Surutilisation du processeur d'une *node* ***

4. **Opérations sur la table COLELE01**

   — Ajout de la colonne *age* ;

   — Mise à jour (initialisation) de la colonne *age* avec la valeur 0 pour tous les enregistrements (55348 tuples affectés) ;

   — Modification du nom de la colonne pour *agemod* ;

   — **Partie 2**

   *** SABOTAGE #2 - Surutilisation de la mémoire d'une *node* ***

   — Suppression d'une colonne (*agemod*) ;

   — Suppression des étudiants au statut D - Décédé (1073 tuples affectés) ;

   — Mise à jour des statuts I - Inactif pour A - Actif (10433 tuples affectés) ;

   — Ajout d'un étudiant (1 tuple affecté).

5. **Opérations sur la table COLBEC01**

   — Mise à jour des notes 59 pour 60 (10064 tuples affectés) ;

   — Suppression des notes des programmes BAS00 et BASE0 (14856 tuples affectés) ;

   — **Partie 3**

   *** SABOTAGE #3 - Saturation de l'espace disque d'une *node* ***

   — Ajout des notes pour l'étudiant créé précédemment en copiant celles d'un autre étudiant (70 tuples affectées) ;

   — Modification du type de la colonne *sessionmeq* ;

   — **Partie 4**

   *** SABOTAGE #4 - Arrêt d'une *node* ***

   — Suppression des colonnes des moyennes (*moyennegroupe* et *moygrou-*

pedec).

6. **Partie 5**

   *** SABOTAGE #5 - Délai réseau d'une *node* ***

7. **Opérations sur la table MDL_USER**

   — Ajout de la colonne *nbrjours* ;

   — Mise à jour de la colonne *nbrjours* par le calcul du nombre de jours passés depuis la dernière connexion de l'étudiant (55561 tuples affectés) ;

   — Suppression des enregistrements des étudiants qui ne se sont pas branchés depuis plus de cinq (5) ans (34954 tuples affectés) ;

   — **Partie 6**

   *** SABOTAGE #6 - Perte de paquets d'une *node* ***

   — Modification du nom de la colonne *nbrannees* ;

   — Mise à jour des enregistrements de la colonne nbrannees (conversion des jours en années) (20607 tuples affectés).

8. **Opérations sur la table MDL_GRADE_GRADES**

   — Suppression des enregistrements dont la note est 0 (721384 tuples affectés) ;

   — Ajout de la colonne *fgarrondie* ;

   — Mise à jour des enregistrements de la colonne *fgarrondie* par l'arrondissement de la colonne *finalgrade* avec deux (2) décimales (514579 tuples affectés).

   — **Partie 7**

   *** SABOTAGE #7 - Réduction de bande passante d'une *node* ***

   — Mise à jour des enregistrements de la colonne *fgarrondie* par le calcul de la moyenne des notes de l'étudiant à l'aide d'une sous-requête et ne conserver qu'une ligne par étudiant (23173 tuples affectés) ;

— Renommer la colonne *fgarrondie* pour *moyenne*.

9. **Opérations sur la table VENPQ_MPULSE_RAPPORT**
   — Suppression des enregistrements pour certaines échelles à l'aide d'une expression régulière (79028 tuples affectés) ;
   — Suppression des enregistrements dont la moyenne des coefficients de risque est 0 à l'aide d'une sous-requête (34552 tuples affectés) ;
   — **Partie 8**

   *** SABOTAGE #8 - Redémarrage d'une *node* avec délai au hasard ***
   — Suppression des enregistrements dont le coefficient de risque est 0 afin de ne pas affecter les moyennes (141430 tuples affectés).
   — **Partie 9**

   *** SABOTAGE #9 - Corruption de paquets d'une *node* ***
   — Calcul des minimums et maximums de chacune des échelles dans une table temporaire (42 tuples affectés) ;
   — **Partie 10**

   *** SABOTAGE #10 - Redémarrage immédiat d'une *node* ***
   — Suppression de la table originale VENPQ_MPULSE_RAPPORT et renommer la table temporaire avec ce même nom ;
10. Réinitialisation des sabotages pouvant potentiellement affecter encore certaines *nodes* ;
11. Extraction des tables vers des fichiers CSV afin de comparer les CRC avec les fichiers originaux de *SQL Server*.

3.5 Sabotages

Différents outils nous permettent de causer les dix (10) sabotages afin de simuler la surutilisation de ressources ou des pannes qui peuvent survenir dans de véritables

centres de données. Nous utilisons Collectl, un logiciel de surveillance versatile développé par Mark Seger, qui fonctionne en service et permet d'emmagasiner les données de performance d'une machine. Grâce à lui, nous pouvons archiver les données de chacune des exécutions du scénario et ainsi observer les effets dévastateurs des sabotages notamment pour des ressources telles que l'utilisation des processeurs, de la mémoire, des disques et du réseau (Helvick, 2008).

L'utilitaire PingInfoView de Nir Sofer est quant à lui utilisé pour examiner la disponibilité des *nodes* pendant le traitement en effectuant une commande Ping à un intervalle régulier (délai de réponse du serveur en millisecondes (ms)). Son journal permet d'analyser plusieurs métriques comme le nombre de paquets perdus au total, le nombre maximum consécutif de paquets perdus ainsi que le minimum / maximum et moyenne du temps du Ping.

3.5.1 Stress-NG

Cet outil, développé par Colin King, permet de mettre un ordinateur au banc d'essai avec une variété impressionnante de tests (King, 2024).

À ce jour, plus de trois cents (300) tests différents permettent de mettre à rude épreuve différentes composantes notamment les processeurs, la mémoire et le système de fichiers.

Surutilisation du processeur (*CPU workload*)

Sabotage #1 : stress-ng -c 1 -l POURCENTAGE -t SECONDES

Ce premier sabotage permet de simuler une utilisation intensive du processeur en utilisant différentes méthodes de façon séquentielle notamment *bitops*, *fibonacci* et *hyperbolic*.

Lors de son utilisation, les variables POURCENTAGE et SECONDES sont géné-

rées de façon aléatoire entre 50 et 79 (%) pour la première et 30 et 119 (s) pour la deuxième.

Surutilisation de la mémoire (*memory workload*)

Sabotage #2 : stress-ng –vm 1 –vm-bytes MEGAOCTETS -t SECONDES

Ce deuxième sabotage permet de simuler une utilisation aggressive de la mémoire en démarrant un processus qui fait en continu des appels mmap(2)/munmap(2) et en écrivant dans la mémoire allouée. Lors de son utilisation, les variables MEGAOCTETS et SECONDES sont générées de façon aléatoire entre 1024 et 2047 (Mo) pour la première et 30 et 119 (s) pour la deuxième.

3.5.2 Fallocate

Saturation (allocation) d'espace disque (*fallocate*)

Ce programme permet de manipuler facilement et rapidement l'allocation d'espace disque pour un fichier. C'est une méthode efficace qui ne génère aucun I/O contrairement à la création d'un fichier en le remplissant de zéros (Canonical, 2019). Il devient alors simple de simuler un manque d'espace disque.

Sabotage #3 : fallocate -l TAILLEG /CHEMIN/FICHIER

Lors de son utilisation, la variable TAILLE est générée de façon aléatoire, soit 6 ou 7 (Go).

3.5.3 Ozone –daemon

Cette commmande du shell Ozone permet de gérer les différents services du système (OM, SCM, RECON et DN).

Arrêt d'une *node*

Sabotage #4 : ozone –daemon stop SERVICE

Ce sabotage nous permet de simuler un serveur souffrant d'un bris matériel ou d'un partitionnement réseau. Ozone va modifier l'état de la *node* pour Obsolète (*Stale*) si cette dernière ne répond pas aux *heartbeats* dans le délai alloué d'une (1) minute. Elle sera considérée Morte (*Dead*) si ce délai dépasse trois (3) minutes (voir le tableau 3.2).

EXTRAIT DE CODE 3.1 – Ozone - Arrêt du service *Datanode*

```
1
2 # LOG SERVOZONE02
3 [shutdown-hook-0] INFO
  ↪ org.apache.hadoop.ozone.HddsDatanodeService: SHUTDOWN_MSG:
4 /************************************************************
5 SHUTDOWN_MSG: Shutting down HddsDatanodeService at
  ↪ servozone02/000.000.000.131
6 ************************************************************/
```

3.5.4 Traffic control (Tc)

Cet utilitaire permet de contrôler le trafic du noyau (*kernel*) Linux avant que ce dernier ne soit distribué vers l'interface réseau, tel que démontré à la figure 3.3. Ce contrôle peut s'effectuer selon plusieurs méthodes telles que le formatage (*Shaping*), la planification (*Scheduling*), les stratégies (*Policing*) et le rejet (*Dropping*).

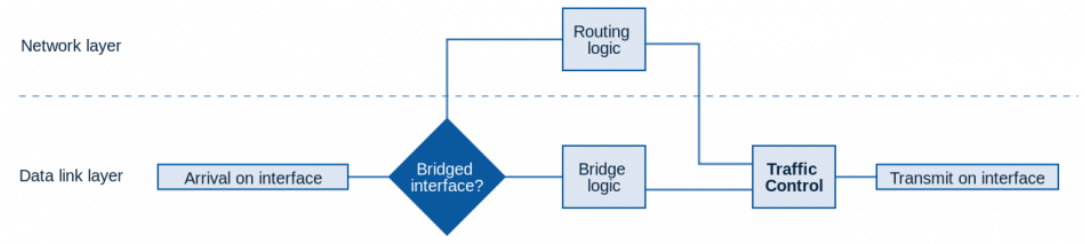

FIGURE 3.3 – Traffic control (EXCENTIS, 2014a).

La base de Tc, c'est ce que l'on appelle la *Queuing Discipline* (qdisc), laquelle représente la politique de planification à une queue (file d'attente). Il existe une panoplie de qdiscs, mais dans notre cas, nous utilisons principalement *Network Emulation* (Netem) et *Token Bucket Filter* (TBF) pour arriver à nos fins. Nous pouvons aussi les combiner ensemble, ce qui permet de tester différents scénarios.

3.5.5 Network emulation (Netem)

Délai réseau (*network delay*)

Le délai réseau est le temps que prend un paquet pour se rendre de sa source à une destination. L'impact de cette problématique cause une latence pour chaque transmission et par conséquent, un ralentissement global qui peut être assez considérable. Ce sabotage nous permet de simuler notamment un serveur qui est géographiquement éloigné. Nous ajoutons le paramètre optionnel de corrélation selon laquelle la prochaine valeur aléatoire dépendra de la dernière. Tel qu'indiqué dans la documentation de Netem, il s'agit d'une approximation et non d'une réelle corrélation statistique (OpenWrt, 2018).

Sabotage #5 : sudo tc qdisc add dev INTERFACE root netem delay LATENCEms VARIATIONms distribution normal

Lors de son utilisation, les variables LATENCE et VARIATION sont générées de façon aléatoire entre 50 et 99 (ms) pour la première et 10 et 19 (ms) pour la deuxième.

Perte de paquets (*packet loss*)

La perte de paquets en réseautique est relativement commune. Elle est souvent causée par la congestion du réseau, des problématiques matérielles ou logicielles et aussi en raison d'un grand nombre de facteurs autres qui peuvent affecter la qualité de la transmission.

La tolérance au pourcentage de paquets perdus diffère selon le type d'application. La retransmission des paquets engendre de la latence et de la variation, ce qui peut occasionner une dégradation du service. Par exemple, la vidéoconférence et la téléphonie VoIP sont moins tolérantes à plus de 1% de paquets perdus. Ce sabotage nous permet de simuler une perte de paquets sur le réseau. En général, une perte de paquets de moins de 3% peut être acceptable. De notre côté, nous utilisons des paramètres au-delà de ce point en plus d'ajouter le facteur de probabilité successive selon le résultat du dernier paquet. Cela permet de simuler du même coup une explosion de paquets perdus (*packet burst losses*) (OpenWrt, 2018).

Sabotage #6 : sudo tc qdisc add dev INTERFACE root netem loss POURCENTAGE PROBABILITÉ

Lors de son utilisation, les variables POURCENTAGE et PROBABILITÉ sont générées de façon aléatoire entre 15 et 25 (%) pour la première et 10 et 19 (%) pour la deuxième.

Corruption de paquets (*packet corruption*)

Tout comme la perte de paquets, la corruption va elle aussi engendrer de la retransmission de paquets et par conséquent, de la latence. Elle s'exécute en modifiant le contenu d'un paquet en fonction du pourcentage demandé. La corruption se voit par une latence et une perte de paquets lors de la commande Ping, mais plus facilement en utilisant l'utilitaire *tcpdump*, qui permet d'analyser tout le trafic réseau en filtrant les paquets qui transitent via l'interface réseau et la *node* de notre choix (Canonical, 2019).

Sabotage #9 : sudo tc qdisc add dev INTERFACE root netem corrupt POURCENTAGE

Lors de son utilisation, la variable POURCENTAGE est générée de façon aléatoire

entre 1 et 10 (%).

3.5.6 Token Bucket Filter (TBF)

Cette commande va limiter la sortie des paquets afin de modifier la vitesse de transfert entre deux machines, comme le présente la figure 3.4.

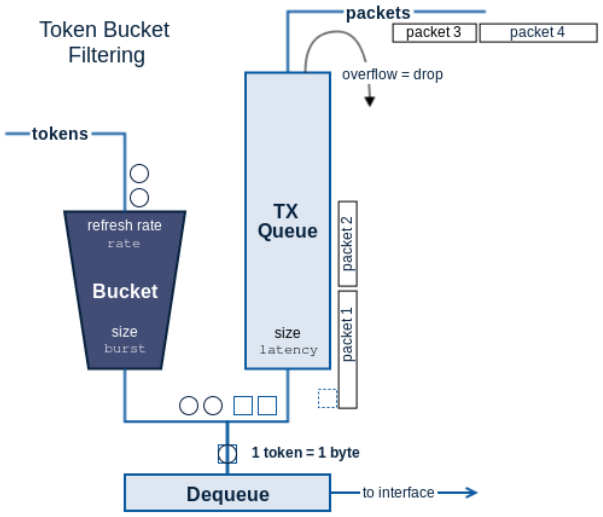

FIGURE 3.4 – Traffic control - Token Bucket Filter (TBF) (EXCENTIS, 2014b).

Réduction de la bande passante

Nous utilisons l'outil iperf qui nous permet de valider l'efficacité du sabotage en calculant la vitesse de transfert entre deux *nodes*. Le fonctionnement est simple : iperf est démarré en mode serveur sur une machine et en mode client sur l'autre.

Sabotage #7 : sudo tc qdisc add dev INTERFACE root tbf rate 128kbit burst 16kbit latency 50ms

3.5.7 Reboot

La commande *reboot* permet d'arrêter ou redémarrer une machine (CANONICAL, 2019). Nous l'utilisons afin de redémarrer une *node* sur le champ mais aussi accompagnée précédemment de la commande *sleep*, afin de retarder le redémarrage.

Redémarrage avec délai d'une *node* au hasard

Sabotage #8 : sleep SECONDES && sudo reboot

Lors de son utilisation, la variable SECONDES est générée de façon aléatoire entre 30 et 119 (s).

Redémarrage immédiat d'une *node* au hasard

Sabotage #10 : sudo reboot

3.6 Scripts

Nous vous présentons les principaux scripts que nous utilisons pour les expérimentations. Ceux-ci sont appelés par le shell, avec ou sans paramètre, afin d'automatiser leur exécution. Plusieurs autres scripts sont disponibles sur notre dépôt GitHub notamment pour l'automatisation de tâches telles que l'initialisation du *cluster* et l'analyse des journaux des différentes *nodes*.

La plupart des scripts utilisent le programme *pdsh* qui permet de lancer une commande en parallèle sur un groupe d'ordinateurs ainsi que *tmux*, un multiplexeur de terminal qui permet de créer et d'accéder à des terminaux à partir d'un même écran (Canonical, 2019). Ce dernier est surtout utilisé afin que le script n'attende pas la fin de l'exécution de la commande, par exemple avec stress-ng :

tmux new -d pdsh -w servozone0[2-7] stress-ng -c 1 -l 80 -t 30s

Note

Les adresses IP, noms d'utilisateurs et mots de passe ont été retirés des sources par question de sécurité.

Script crud.sh

Il s'agit du script principal du projet. Il permet de lancer le test de façon auto-

nome en lui spécifiant les paramètres désirés. Ce dernier peut s'exécuter en trois (3) modes : Opérations, Sabotages ou Test (sabotages seulement sans aucune opération CRUD).

Le déroulement du script :

- Création de l'arborescence pour la récolte de données ;
- Sélection aléatoire des *nodes* pour chacun des sabotages et génération des variables aux valeurs aléatoires (s'il y a lieu) ;
- Lancement du client Hive et du script HQL associé ;
- Génération des journaux avant et après (*pipeline*s, *container*s et *nodes*) ;
- Génération des journaux du déroulement et de fin de traitement (Hive, SCM, OM, *nodes*, *container*s, *key*s, journal d'exécution et validation des fichiers) ;
- Sauvegarde des journaux du service Collectl.

Script validation.sh

Ce script est lancé à partir de crud.sh après l'extraction des tables. Son mandat est de vérifier les sommes de contrôle (CRC) de chacunes d'elles afin de corroborer la cohérence des données des fichiers générés comparativement aux fichiers originaux.

Chapitre 4

RÉSULTATS

L'objectif principal de ce mémoire étant d'évaluer la fiabilité et la cohérence d'Ozone, nous vous présentons dans ce chapitre les résultats obtenus lors de nos expérimentations effectuées sur un *cluster* de sept (7) *nodes* en comparant les données obtenues en mode Opérations à celles recueillies en mode Sabotages pour les dix (10) parties du scénario lors de nos cents (100) itérations. Nous aborderons d'abord l'état du *cluster* à la fin de nos simulations et, par la suite, les résultats sommaires vous seront présentés. Enfin, nous allons forer plus en détails les comparaisons observées entre les divers sabotages utilisés.

4.1 État du *cluster*

Au tout début, une fois l'ingestion initiale de nos principaux fichiers, nous avions certaines valeurs de départ notamment le nombre de *container*s, le nombre de *key*s et la taille (espace disque utilisé), tel que nous pouvons le voir au tableau 4.1.

Chaque fois que nous avons lancé le script crud.sh, une série d'opérations CRUD ont été effectuées sur les tables gérées stockées dans Ozone, générant ainsi une panoplie de *key*s (répertoires et fichiers) lors des commandes HQL de Hive traduites

TABLEAU 4.1 – Valeurs à l'initialisation du *cluster*

Attribut	Valeur
Séquence	0
Taille	2.3 Go
Conteneurs	5
Nombre de clés	11

sous forme de tâches *MapReduce*. Ce sont cent dix (110) nouvelles *key*s supplémentaires créées dont la taille varie de quelques octets à plusieurs mégaoctets, tel que démontré à la figure 4.1.

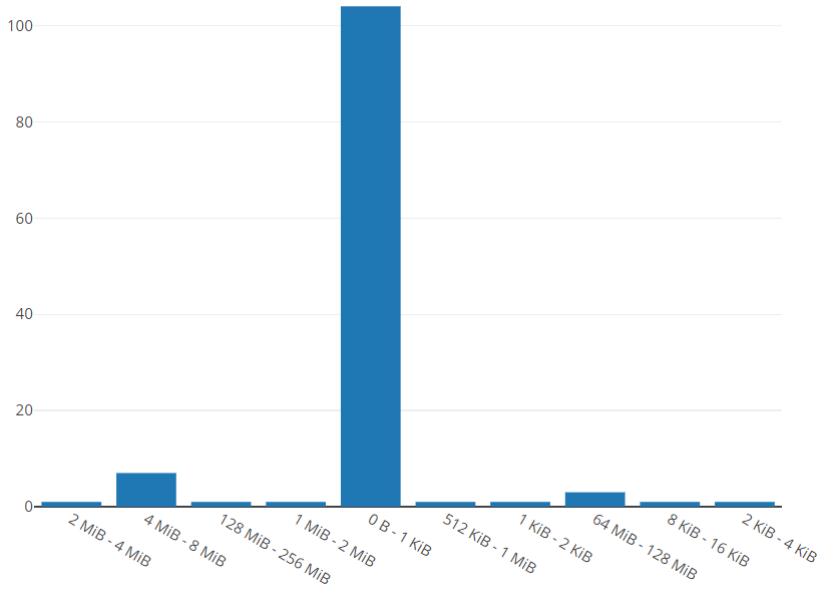

FIGURE 4.1 – Distribution de la taille des fichiers

Nous apercevons à la figure 4.2 que l'accroissement des *key*s demeure stable et inchangé et que l'augmentation du nombre de *container*s est relativement linéraire à première vue avec une moyenne de 2.79 en mode Opérations et de 3.71 en mode Sabotages, comme nous pouvoir le voir à la figure 4.3.

La taille des données quant à elle, varie grandement, mais tout en conservant une croissance soutenue comme le démontre la figure 4.4. Cette croissance en continu est reliée à la variation de l'espace disque des *nodes*, car comme nous le voyons aussi à la figure 4.5, l'espace disque utilisé est toujours croissant. Cette métrique

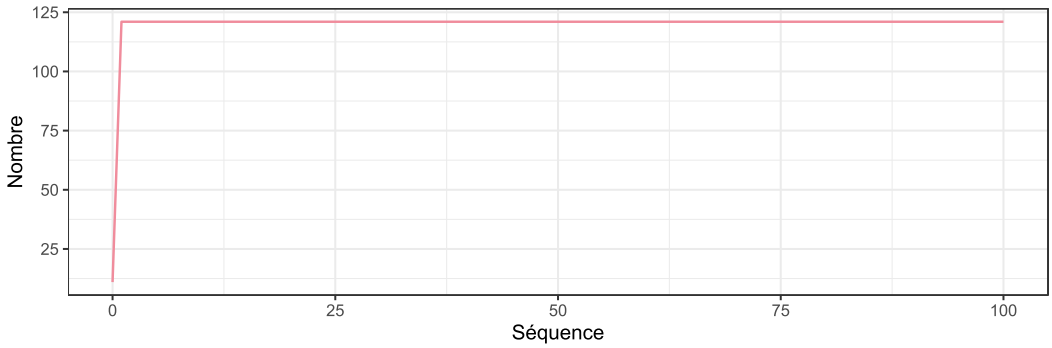

FIGURE 4.2 – Accroissement du nombre de *key*s

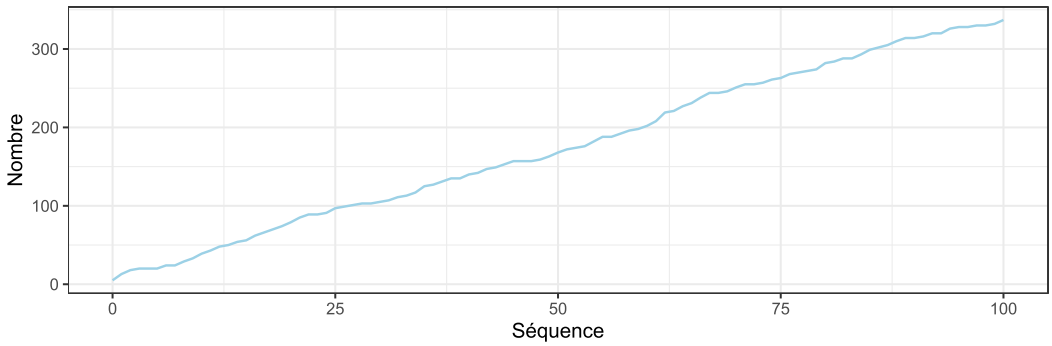

FIGURE 4.3 – Augmentation du nombre de *container*s

varie considérablement selon plusieurs facteurs. Ces fluctuations peuvent être engendrées par les sabotages. Par exemple, lorsqu'un *pipeline* est détruit et remplacé par un nouveau, la ou les DNs laissées de côté devraient normalement retrouver leur espace disque suite au travail du RM de façon asynchrone. De plus, le Jira du projet Ozone fait état de quatre (4) problèmes en lien avec des fichiers orphelins dont trois (3) sont résolus dans la version 1.4.0 (HDDS-7156, HDDS-7592 et HDDS-8139) et un autre (1) est toujours ouvert (HDDS-7728).

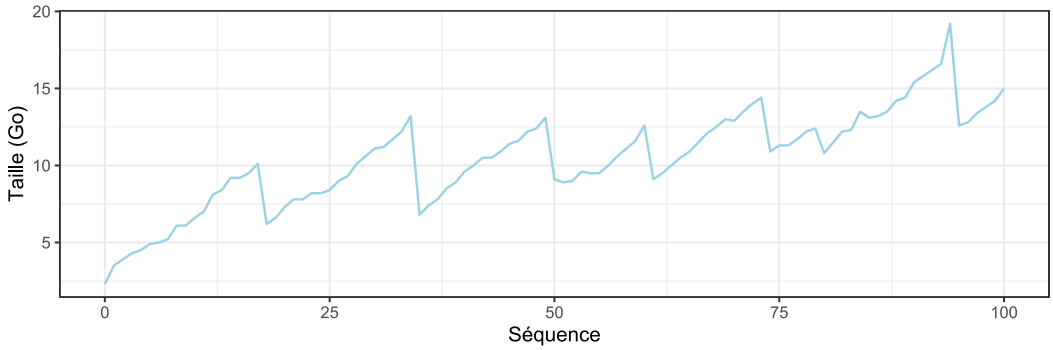

FIGURE 4.4 – Variation de la taille des données

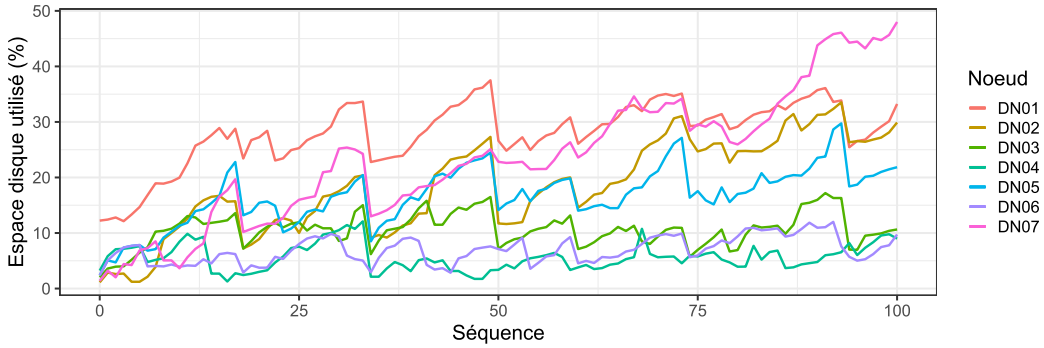

FIGURE 4.5 – Variation de l'espace disque des *nodes*

4.2 Résultats généraux

Sommairement, tel que vu dans le tableau 4.2, le mode Opérations (O) a démontré une fiabilité et une cohérence parfaites avec une durée d'exécution médiane de 8 min 6 s (μ 8 min 10 s). Le mode Sabotages (S) a quant à lui mis en lumière une petite déficience au niveau de la fiabilité du protocole RAFT implémenté par la librairie Ratis tout en conservant une cohérence sans reproche. Il faut dire que le but de ce mode était justement de tester les limites du système distribué et cela se remarque par une durée d'exécution médiane de 20 min 50 s (μ 25 min 6 s), soit 2.57 fois plus longue que le mode Opérations. La figure 4.6 nous montre bien que l'étendue de la durée est majeure et que la dispersion des durées est beaucoup moins condensée que le mode Opérations, tel que vu à la figure 4.7.

TABLEAU 4.2 – Résultats

Mode	Exéc.	Fiabilité	Cohérence	Durée (μ)	Durée (med)	E
Opérations (O)	50	100%	100%	486 s	489.90 s	199
Sabotages (S)	50	90%	100%	1250 s	1506.61 s	7751

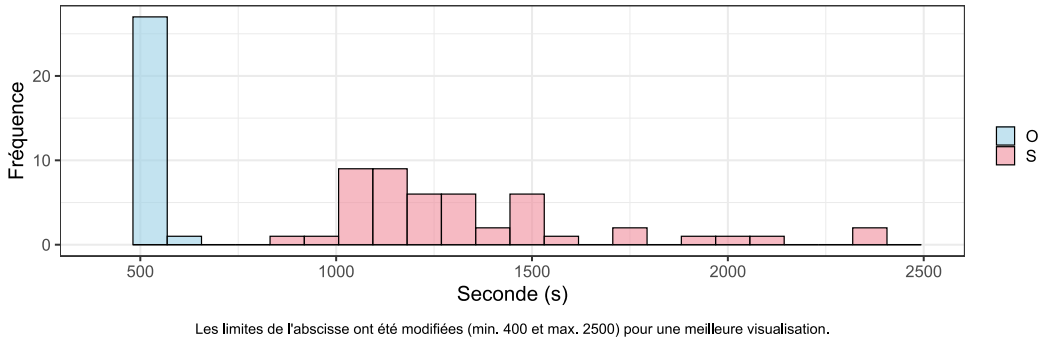

FIGURE 4.6 – Fréquence de la durée de traitement

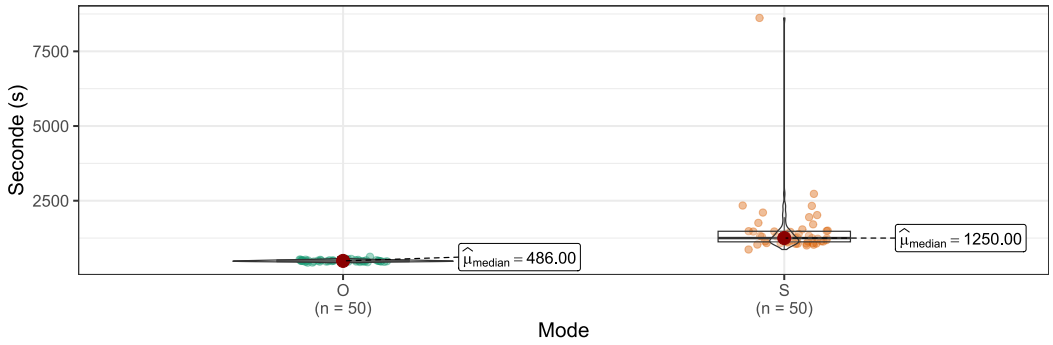

FIGURE 4.7 – Fréquence de la durée de traitement

4.2.1 Fiabilité

Cet aspect a été mis à rude épreuve lors de nos expérimentations. En effet, le mode Sabotages a souffert de défaillances à cinq (5) reprises (10%) contrairement au mode Opérations qui s'en est tiré avec brio.

Bien que ces erreurs n'eussent pas remis en cause la disponibilité et la cohérence du système sur le coup, une dégradation de la performance aurait été perceptible à long terme. À chaque fois, le même scénario se répétait pour les exécutions #50,

#60, #78, #88 et #94 : il n'y avait plus d'espace disponible sur une *node* d'un *pipeline* car elle était affectée par le sabotage #3 (saturation de l'espace disque par la commande *fallocate*) et cela provoquait des erreurs d'écriture. Par la suite, même après un redémarrage de la *node* en question, le service de *Datanode* n'était plus capable de rejoindre le *cluster*, comme le montre l'extrait de code 4.1).

La solution pour rétablir le tout était d'arrêter le service *Datanode*, de supprimer le répertoire Ratis et de redémarrer le service *Datanode*. Sans la suppression du répertoire Ratis, la *node* était bien active mais ne pouvait plus joindre aucun *pipeline*. Après vérification au Jira du projet Ratis, ce problème a été corrigé dans la version 3.0.0 (RATIS-761).

L'espace disque est crucial pour un système distribué. Comme chaque *node* de notre *cluster* ne dispose que d'un disque de 10 Go, dont 498.91 Mo sont réservés, la marge de manoeuvre des 9.51 Go disponibles est mince et peut rapidement devenir critique. Cela a été le cas pour l'exécution #60 avec l'ajout d'un fichier de 7 Go sur le disque sachant que ce dernier était déjà utilisé à 2.44 Go, soit 26.34% de la capacité totale réservée à Ozone.

Note

Lors de nos tests, le service *Container Balancer* n'était pas activé. Ce dernier permet d'équilibrer l'utilisation d'espace disque des DNs en déplaçant des *containers* de DNs surtilisées vers d'autres qui sont sous-utilisées.

EXTRAIT DE CODE 4.1 – Erreur Ratis - Mode sabotages - Séquence #60

```
1
2 # LOG DU SCM
```

```
[EventQueue-PipelineActionsForPipelineActionHandler] INFO
↪ PipelineActionHandler: Received pipeline action CLOSE for
↪ PipelineID=c1bb2388-9022-457e-9d5e-03ac8fb99475 from datanode
↪ b24bcb3c-7b0d-4564-a324-07315c38939d. Reason : YAMLException:
↪ java.io.IOException: No space left on device

# LOG DE SERVOZONE07
[ChunkWriter-9-0] ERROR ContainerStateMachine: group-03AC8FB99475:
↪ writeChunk writeStateMachineData failed: blockIdcontainerID:
↪ 3017
localID: 111677748019206709
blockCommitSequenceId: 0
replicaIndex: 0
logIndex 429 chunkName 111677748019206709_chunk_1
YAMLException: java.io.IOException: No space left on device

[Datanode State Machine Daemon Thread] ERROR
RunningDatanodeState: Error in executing end point task.
```

4.2.2 Cohérence

De ce côté, nos résultats sont sans équivoque : tous les CRC sont conformes et identiques, tel que vu au tableau 4.2. Bien que nos sabotages aient mis en lumière les problèmes causés par l'espace disque insuffisant et la présence de fichiers orphelins provoquant l'accroissement de l'espace disque utilisé, le système a pu continuer à fonctionner normalement et donner des résultats cohérents.

4.2.3 Sabotages

Les moyens utilisés afin de simuler les défaillances ont bien démontré leur efficacité au tableau 4.2, par la comparaison de la durée médiane du temps d'exécution en mode Sabotages, qui s'est avérée beaucoup plus longue qu'en mode Opérations. Nous avons bombardé Ozone de cinq cents (500) sabotages qui ont été dispersés sur les sept (7) *nodes* du *cluster*.

Comme le démontre la figure 4.8, l'assignation aléatoire des sabotages aux *nodes* a malgré tout généré une distribution assez uniforme de ceux-ci. Les données brutes utilisées pour la compilation des sabotages peuvent être consultées au tableau A.4 en Annexe A.

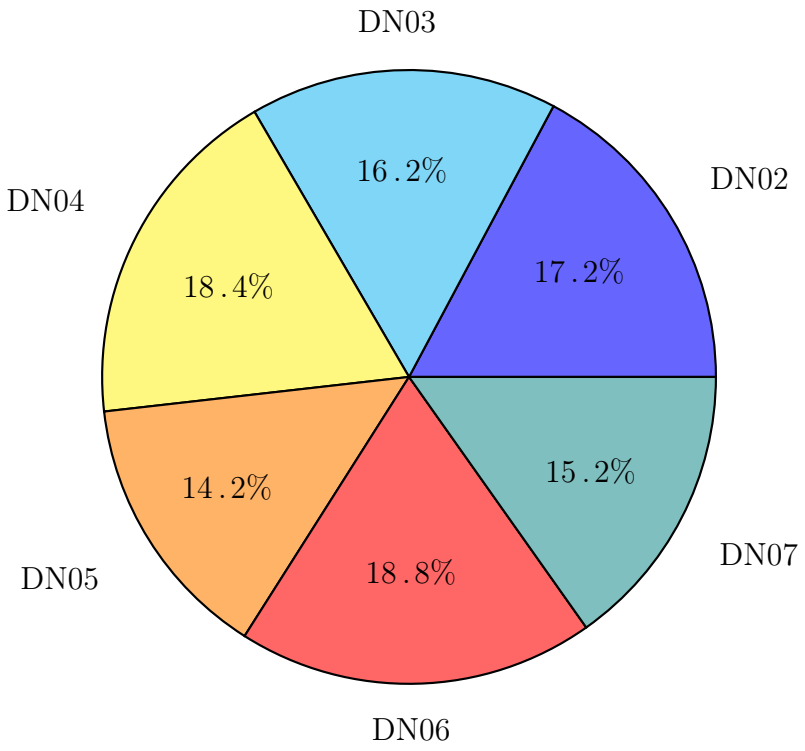

FIGURE 4.8 – Distribution des sabotages

La compilation des journaux d'éxécution du script principal avec l'activation des sabotages permet de bien visualiser l'affectation de ces derniers sur les *nodes* du *cluster* (tableau A.5). Les données brutes des éxécutions du script peuvent être

elles aussi consultées en Annexe A.

4.3 Résultats spécifiques aux parties du scénario

Les résultats présentés au tableau 4.2 sont indéniables, les moyennes (arithmétique et médiane) de la durée d'exécution varient selon le mode Opérations (O) ou Sabotages (S). Afin d'évaluer quels sabotages semblent avoir le plus d'effets sur la performance d'Ozone, nous évaluons d'abord les résultats obtenus en vérifiant si les données représentent une distribution normale ou non (Shapiro-Wilk) pour ensuite effectuer un second test dans le but de valider si la distribution de nos deux groupes (Opérations et Sabotages) est statistiquement significative ou non (Wilcoxon-Mann-Whitney). Nous allons utiliser le même *modus operandi* pour la validation des dix (10) sabotages.

Le test Shapiro-Wilk est un test de normalité. Il est couramment utilisé et performe très bien pour des échantillons dont la taille varie entre trente (30) et cinquante (50). Basé sur l'hypothèse nulle (H0) que les données sont normalement distribuées, le résultat de *p-value* plus grand que 0.5 permet d'assumer que la distribution semble normale ou de la rejeter lorsque la valeur est plus petite que 0.5 (Ahad et al., 2011).

Le test Wilcoxon-Mann-Whitney (Test U de Mann-Whitney ou test de la somme des rangs Wilcoxon) est utilisé pour déterminer s'il y a des différences statistiquement significatives entre deux groupes indépendants. Il est non paramétrique, c'est-à-dire qu'il n'assume pas que la distribution est normale. Basé sur l'hypothèse nulle (H0) qu'il n'y a pas de différence entre les deux groupes, le résultat de *p-value* plus grand que 0.5 permet d'assumer qu'il n'y a pas de différence, en terme de tendance centrale entre ces derniers. Une valeur plus petite que 0.5 permet d'assumer l'hypothèse alternative qu'il y a une différence entre les deux

groupes (NACHAR, 2008).

4.3.1 Partie 1 - Surutilisation du processeur

Pour ce premier sabotage, deux métriques (variables du script) aléatoires venaient altérer les effets sur la *node* affectée. Le pourcentage d'utilisation du processeur ainsi que la durée en secondes. La figure 4.9 démontre que les *nodes* ont souffert d'une utilisation du processeur comprise entre 50% et 78% (μ 62.56%) pour une durée comprise entre 32 s et 116 s (μ 70.44 s).

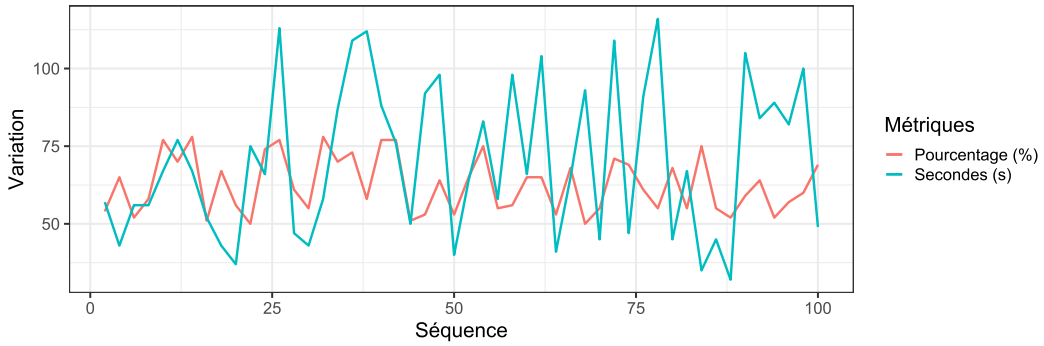

FIGURE 4.9 – Partie 1 - Métriques de la surutilisation du processeur

Les données récoltées lors de l'exécution de cette partie du script nous démontrent qu'à première vue, en visualisant la figure 4.10, les distributions des temps d'exécution en mode Opérations (O) et en mode Sabotages (S) semblent relativement normales.

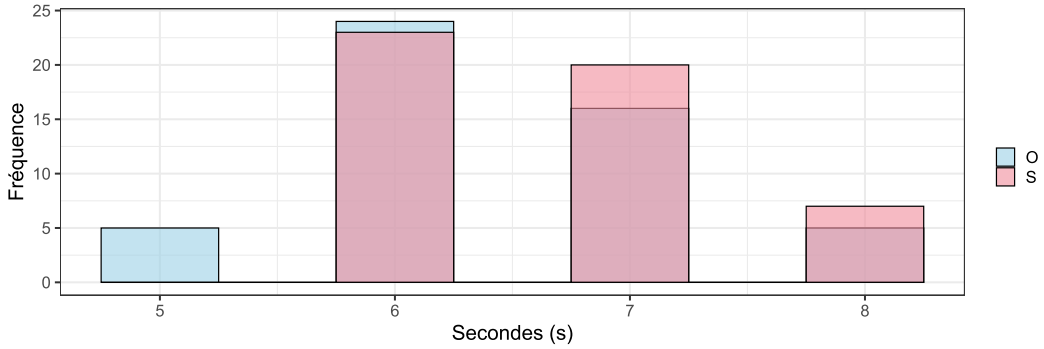

FIGURE 4.10 – Partie 1 - Fréquence de la durée de traitement

Le résultat du test Shapiro-Wilk sur chacune des distributions nous permet de constater qu'en fait, les deux ne le sont pas. Il est de 2.45e-05 pour le mode Opérations et de 2.113e-07 pour le mode Sabotages.

Comme les deux distributions ne sont pas normales, nous pouvons effectuer le test U de Mann–Whitney (non paramétrique) pour comparer les deux groupes de données. Avec un résultat de 0.11000, tel que vu à la figure 4.11, la différence avec respect de la variable durée n'est pas statistiquement significative et nous pouvons donc conclure que ce sabotage ne semble pas avoir eu de réel effet sur la performance d'Ozone.

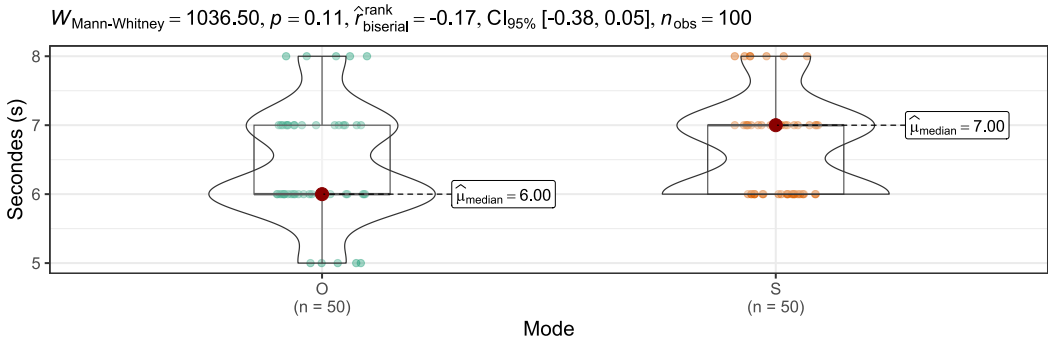

FIGURE 4.11 – Partie 1 - Comparaison de la durée de traitement

4.3.2 Partie 2 - Surutilisation de la mémoire

Pour ce deuxième sabotage, deux métriques (variables du script) aléatoires venaient altérer les effets sur la *node* affectée. Le nombre de mégaoctets (Mo) supplémentaires d'utilisation de la mémoire ainsi que la durée en secondes. La figure 4.12 démontre que les *nodes* ont souffert d'une utilisation moyenne de la mémoire comprise entre 1026 Mo et 2027 Mo (μ 1492 Mo), soit 36.85% de la mémoire disponible et d'une durée comprise entre 30 s et 119 s (μ 72.44 s).

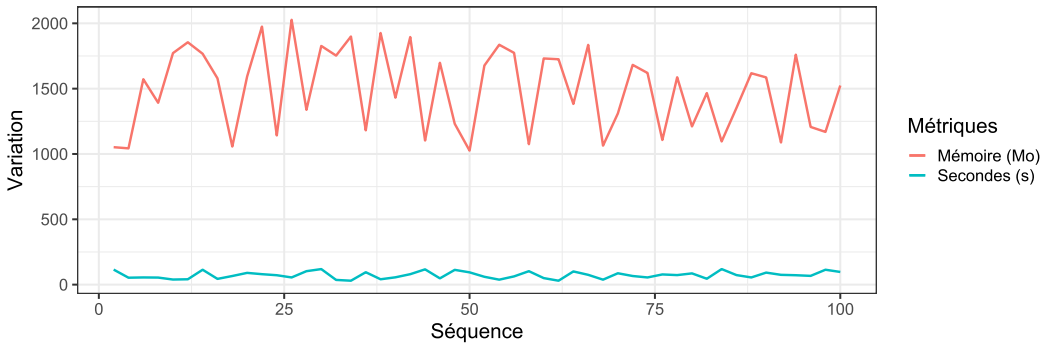

FIGURE 4.12 – Partie 2 - Métriques de la surutilisation de la mémoire

Les données récoltées lors de l'exécution de cette partie du script nous démontrent qu'à première vue, en visualisant la figure 4.13, les distributions des temps d'exécution en mode Opérations (O) et en mode Sabotages (S) ne semblent pas normales.

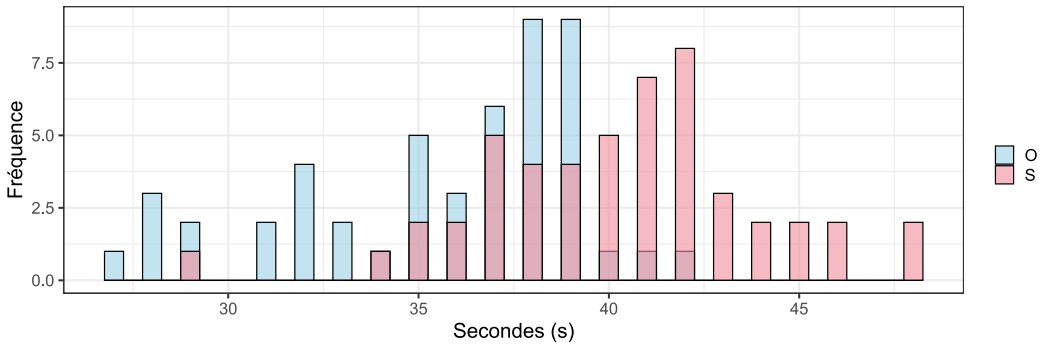

FIGURE 4.13 – Partie 2 - Fréquence de la durée de traitement

Le résultat du test Shapiro-Wilk sur chacune des distributions nous permet de constater que celle du mode Opérations n'est pas normale avec une valeur de 0.00109 contrairement à 0.38290 pour le mode Sabotages, indiquant que cette dernière l'est.

Comme au moins une des deux distributions n'est pas normale, nous pouvons quand même effectuer le test U de Mann–Whitney (non paramétrique) pour vérifier les deux groupes de données. Avec un résultat de 2.633e-08, tel que vu à

la figure 4.14, la différence avec respect de la variable durée est statistiquement significative, nous pouvons donc conclure que ce sabotage a eu un certain effet sur la performance d'Ozone.

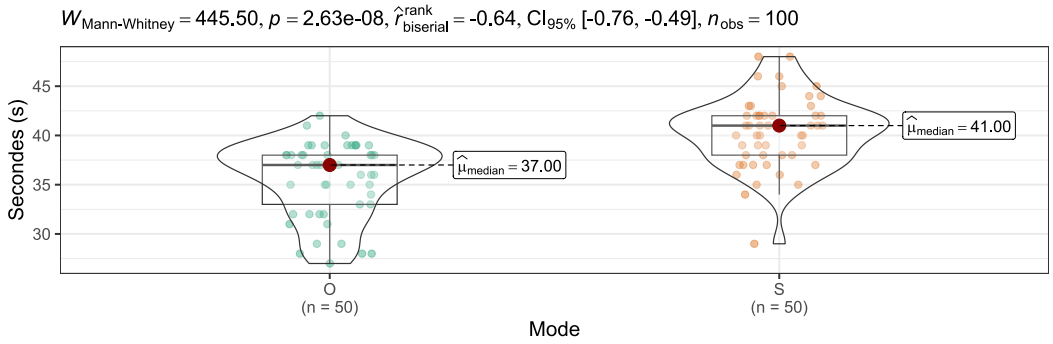

FIGURE 4.14 – Partie 2 - Comparaison de la durée de traitement

4.3.3 Partie 3 - Saturation de l'espace disque

Pour ce troisième sabotage, une seule métrique (variable du script) aléatoire venait altérer les effets sur la *node* affectée. Le nombre de gigaoctets (Go) supplémentaires d'espace disque utilisé. La figure 4.15 démontre que les *nodes* ont souffert d'une utilisation d'espace disque supplémentaire comprise entre 6 Go et 7 Go pour une moyenne réelle de 6.48 Go, soit 68.21% de la taille totale du disque.

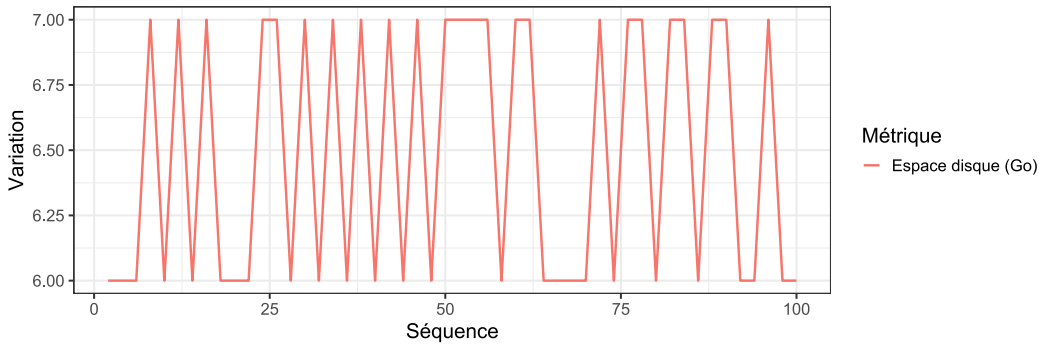

FIGURE 4.15 – Partie 3 - Métrique de la saturation de l'espace disque

Les données récoltées lors de l'exécution de cette partie du script nous démontrent qu'à première vue, en visualisant la figure 4.16, la distribution des temps d'exé-

cution en mode Opérations (O) semble relativement normale tandis que celle en mode Sabotages (S) ne semble pas l'être.

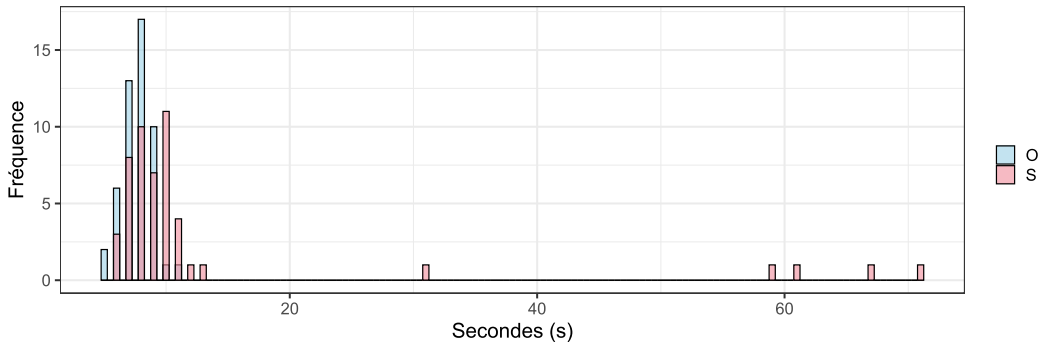

FIGURE 4.16 – Partie 3 - Fréquence de la durée de traitement

Le résultat du test Shapiro-Wilk sur chacune des distributions nous permet de constater qu'en fait, les deux ne le sont pas. Il est de 0.01160 pour le mode Opérations et de 1.481e-12 pour le mode Sabotages.

Comme les deux distributions ne sont pas normales, nous pouvons effectuer le test U de Mann–Whitney (non paramétrique) pour vérifier les deux groupes de données. Avec un résultat de 8.905e-05, tel que vu à la figure 4.17, la différence avec respect de la variable durée est statistiquement significative, nous pouvons donc conclure que ce sabotage a eu un certain effet sur la performance d'Ozone.

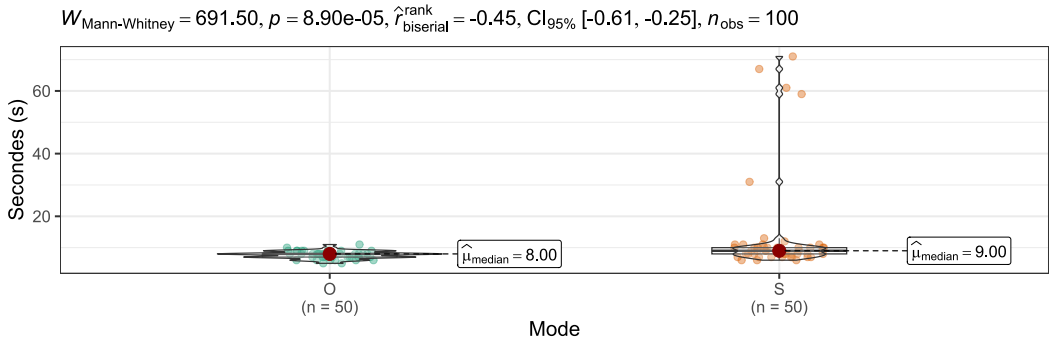

FIGURE 4.17 – Partie 3 - Comparaison de la durée de traitement

4.3.4 Partie 4 - Arrêt d'une *node*

Pour ce quatrième sabotage, aucune métrique (variable du script) ne venait altérer les effets sur la *node* affectée. Le service DN demeurait arrêté jusqu'à la fin de l'exécution, sauf si la *node* était redémarrée lors des sabotages #8 ou #10.

Les données récoltées lors de l'exécution de cette partie du script nous démontrent qu'à première vue, en visualisant la figure 4.18, la distribution des temps d'exécution en mode Opérations (O) semble relativement normale tandis que celle en mode Sabotages (S) ne semble pas l'être.

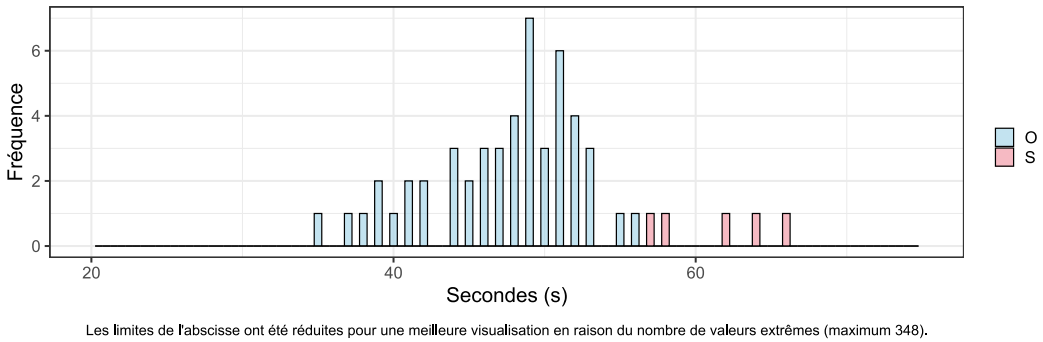

FIGURE 4.18 – Partie 4 - Fréquence de la durée de traitement

Le résultat du test Shapiro-Wilk sur chacune des distributions nous permet de constater qu'en fait, les deux ne le sont pas. Il est de 0.03435 pour le mode Opérations et de 0.00022 pour le mode Sabotages.

Comme les deux distributions ne sont pas normales, nous pouvons effectuer le test U de Mann–Whitney (non paramétrique) pour vérifier les deux groupes de données. Avec un résultat de 2.2e-16, tel que vu à la figure 4.19, la différence avec respect de la variable durée est statistiquement significative, nous pouvons donc conclure que ce sabotage a eu un certain effet sur la performance d'Ozone.

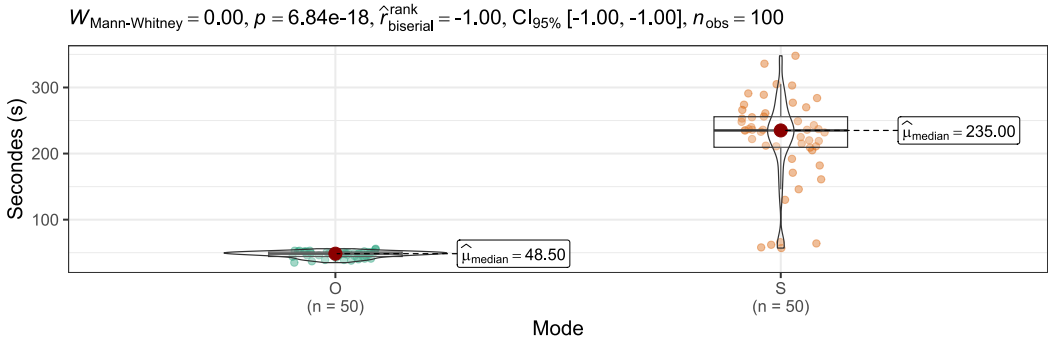

FIGURE 4.19 – Partie 4 - Comparaison de la durée de traitement

4.3.5 Partie 5 - Délai réseau

Pour ce cinquième sabotage, deux métriques (variables du script) venaient altérer les effets sur la *node* affectée. Le nombre de millisecondes (ms) de délai réseau plus ou moins la valeur de l'intervalle. La figure 4.20 démontre que les *nodes* ont souffert d'un délai réseau compris entre 53 ms et 99 ms (μ 76.98 ms) plus ou moins ($\pm$) un intervalle compris entre 10 ms et 19 ms (μ 14.34 ms).

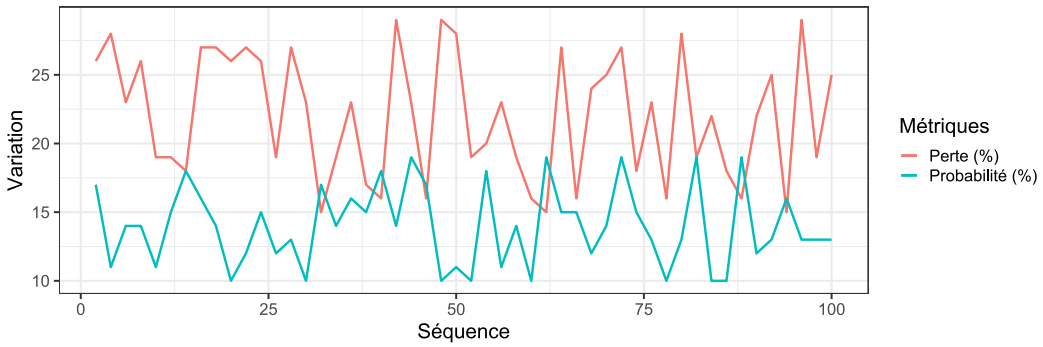

FIGURE 4.20 – Partie 5 - Métriques du délai réseau

Les données récoltées lors de l'exécution de cette partie du script nous démontrent qu'à première vue, en visualisant la figure 4.21, la distribution des temps d'exécution en mode Opérations (O) semble relativement normale tandis que celle en mode Sabotages (S) ne semble pas l'être.

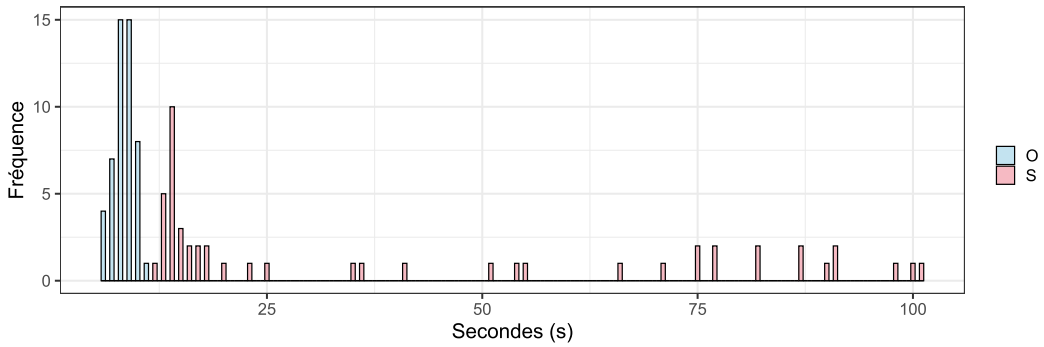

FIGURE 4.21 – Partie 5 - Fréquence de la durée de traitement

Le résultat du test Shapiro-Wilk sur chacune des distributions nous permet de constater qu'en fait, les deux ne le sont pas. Il est de 0.00603 pour le mode Opérations et de 3.775e-07 pour le mode Sabotages.

Comme les deux distributions ne sont pas normales, nous pouvons effectuer le test U de Mann–Whitney (non paramétrique) pour vérifier les deux groupes de données. Avec un résultat de 5.06e-18, tel que vu à la figure 4.22, la différence avec respect de la variable durée est statistiquement significative, nous pouvons donc conclure que ce sabotage a eu un certain effet sur la performance d'Ozone.

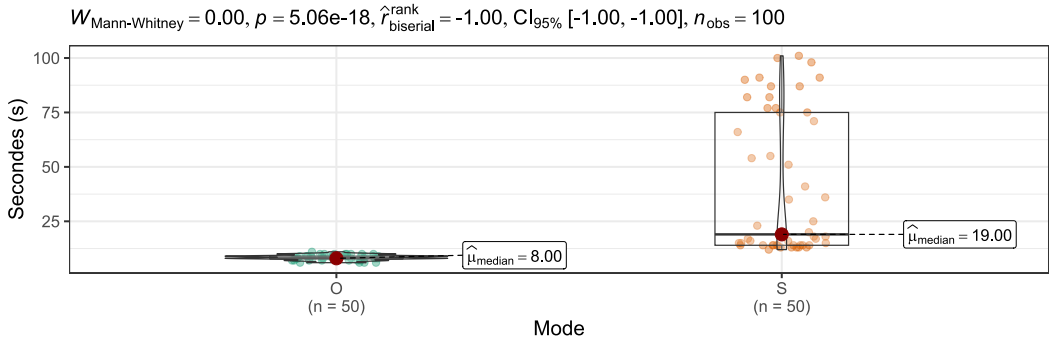

FIGURE 4.22 – Partie 5 - Comparaison de la durée de traitement

4.3.6 Partie 6 - Perte de paquets

Pour ce sixième sabotage, deux métriques (variables du script) aléatoires venaient altérer les effets sur la *node* affectée. Le pourcentage de paquets perdus et la

probabilité successive selon la valeur du dernier. La figure 4.23 démontre que les *nodes* ont souffert d'une perte de paquets comprise entre 15% et 29% (μ 22.04%) plus ou moins ($\pm$) une probabilité de simuler un *packet burst losses* comprise entre 10% et 19%, soit une moyenne de 13.98%.

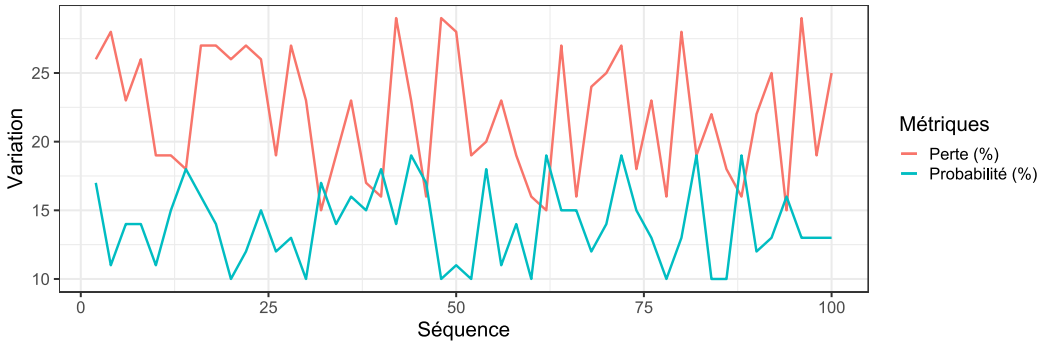

FIGURE 4.23 – Partie 6 - Métriques de la perte de paquets

Les données récoltées lors de l'exécution de cette partie du script nous démontrent qu'à première vue, en visualisant la figure 4.24, la distribution des temps d'exécution en mode Opérations (O) semble relativement normale tandis que celle en mode Sabotages (S) ne semble pas l'être.

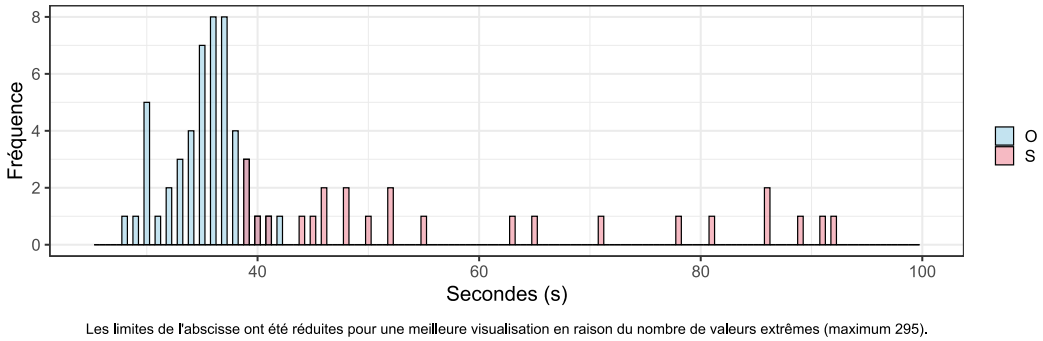

FIGURE 4.24 – Partie 6 - Fréquence de la durée de traitement

Le résultat du test Shapiro-Wilk sur chacune des distributions nous permet de constater que celle du mode Opérations est normale avec une valeur de 0.14660 contrairement à 0.00061 pour le mode Sabotages, indiquant que cette dernière ne l'est pas.

Comme au moins une des deux distributions n'est pas normale, nous pouvons effectuer le test U de Mann–Whitney (non paramétrique) pour vérifier les deux groupes de données. Avec un résultat de 1.88e-17, tel que vu à la figure 4.25, la différence avec respect de la variable durée est statistiquement significative, nous pouvons donc conclure que ce sabotage a eu un certain effet sur la performance d'Ozone.

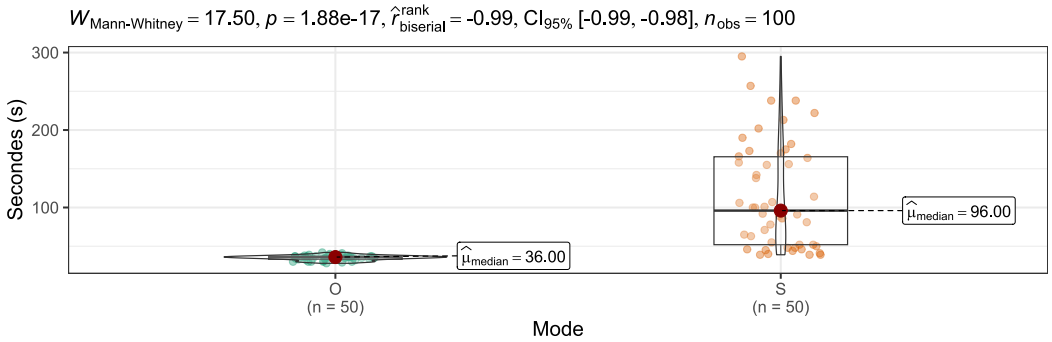

FIGURE 4.25 – Partie 6 - Comparaison de la durée de traitement

4.3.7 Partie 7 - Réduction de bande passante

Pour ce septième sabotage, aucune métrique (variable du script) ne venait altérer les effets sur la *node* affectée. La réduction était constante à 128 Mbit/sec au lieu de 19 Gbit/sec en temps normal, soit cent cinquante (150) fois moins.

Les données récoltées lors de l'exécution de cette partie du script nous démontrent qu'à première vue, en visualisant la figure 4.26, la distribution des temps d'exécution en mode Opérations (O) semble relativement normale tandis que celle en mode Sabotages (S) ne semble pas l'être.

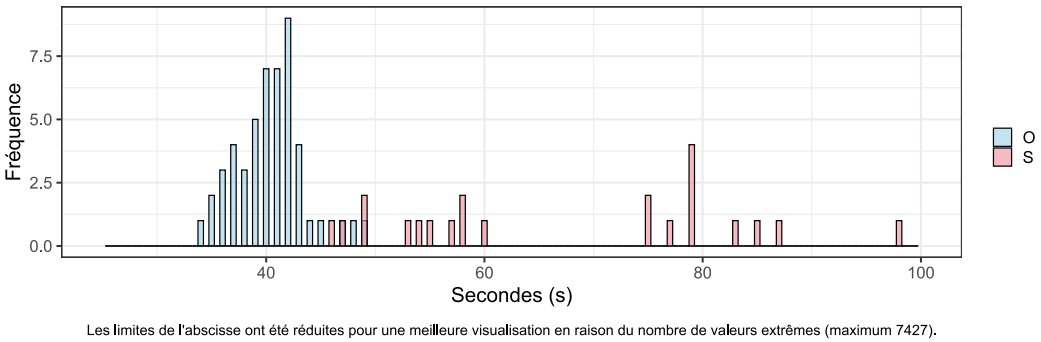

FIGURE 4.26 – Partie 7 - Fréquence de la durée de traitement

Le résultat du test Shapiro-Wilk sur chacune des distributions nous permet de constater que celle du mode Opérations est normale avec une valeur de 0.13510 contrairement à 6.594e-14 pour le mode Sabotages, indiquant que cette dernière ne l'est pas.

Comme au moins une des deux distributions n'est pas normale, nous pouvons effectuer le test U de Mann–Whitney (non paramétrique) pour vérifier les deux groupes de données. Avec un résultat de 9.76e-18, tel que vu à la figure 4.27, la différence avec respect de la variable durée est statistiquement significative, nous pouvons donc conclure que ce sabotage a eu un certain effet sur la performance d'Ozone.

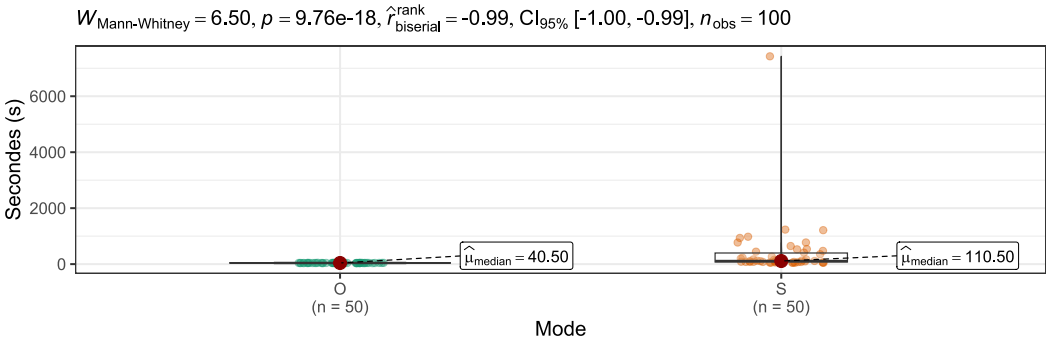

FIGURE 4.27 – Partie 7 - Comparaison de la durée de traitement

4.3.8 Partie 8 - Redémarrage d'une *node* avec délai au hasard

Pour ce huitième sabotage, une seule métrique (variable du script) aléatoire venait altérer les effets sur la *node* affectée. Le délai en secondes avant le redémarrage. La figure 4.28 démontre que les *nodes* ont souffert d'un redémarrage à retardement dont le délai est compris entre 31 s et 116 s pour une moyenne de 79.02 s.

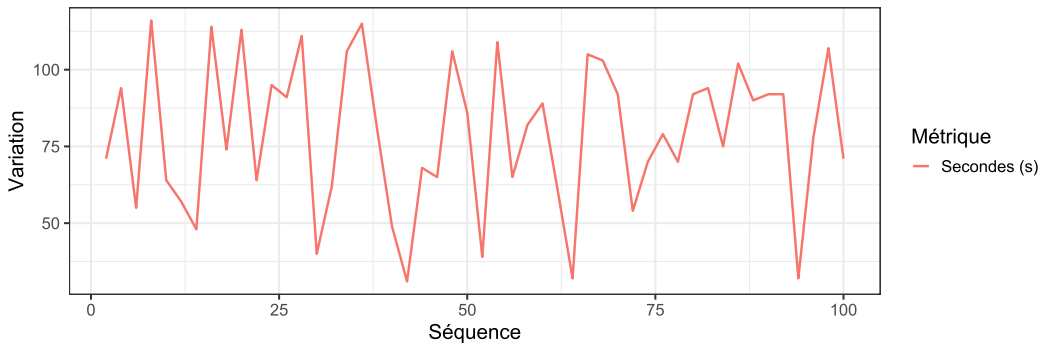

FIGURE 4.28 – Partie 8 - Redémarrage d'une *node* avec délai au hasard

Les données récoltées lors de l'exécution de cette partie du script nous démontrent qu'à première vue, en visualisant la figure 4.29, la distribution des temps d'exécution en mode Opérations (O) en mode Sabotages (S) ne semblent pas normales.

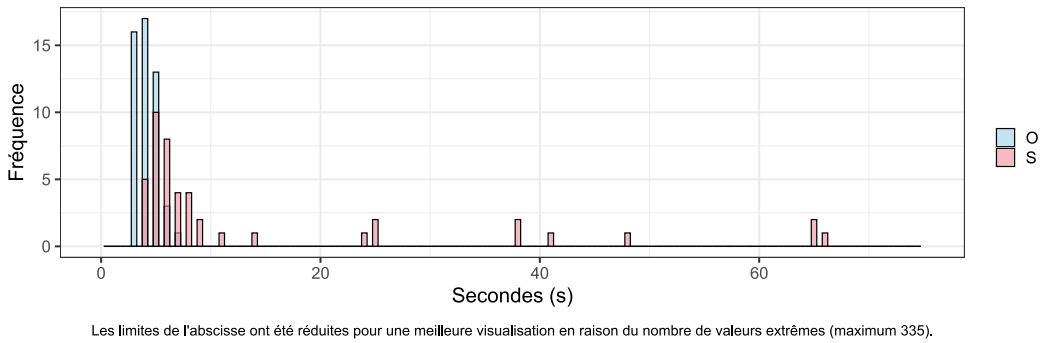

FIGURE 4.29 – Partie 8 - Fréquence de la durée de traitement

Le résultat du test Shapiro-Wilk sur chacune des distributions nous permet de constater qu'effectivement, les deux ne le sont pas. Il est de 3.864e-05 pour le

mode Opérations et de 5.677e-12 pour le mode Sabotages.

Comme les deux distributions ne sont pas normales, nous pouvons effectuer le test U de Mann–Whitney (non paramétrique) pour vérifier les deux groupes de données. Avec un résultat de 3.432e-12, tel que vu à la figure 4.30, la différence avec respect de la variable durée est statistiquement significative, nous pouvons donc conclure que ce sabotage a eu un certain effet sur la performance d'Ozone.

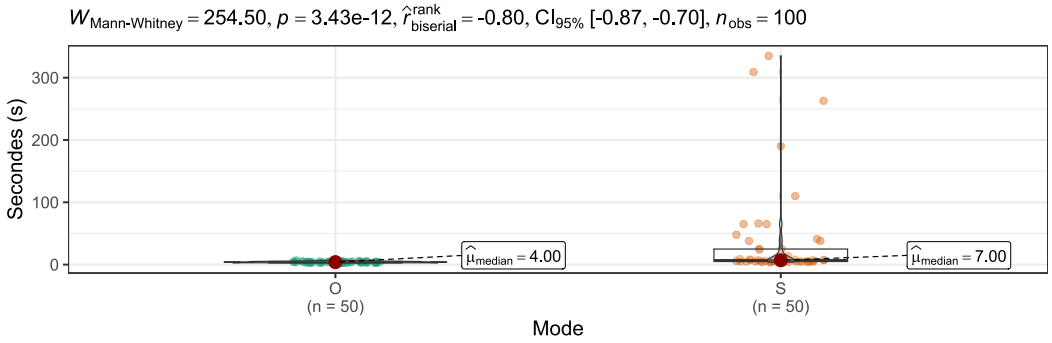

FIGURE 4.30 – Partie 8 - Comparaison de la durée de traitement

4.3.9 Partie 9 - Corruption de paquets

Pour ce neuvième sabotage, une seule métrique (variable du script) aléatoire venait altérer les effets sur la *node* affectée. Le pourcentage de paquets corrompus. La figure 4.31 démontre que les *nodes* ont souffert d'une corruption de paquets comprise entre 1% et 10% pour une moyenne de μ 5.56%.

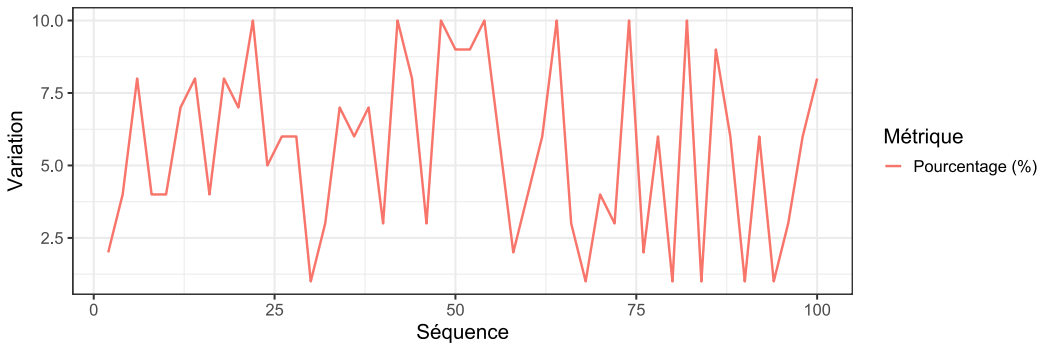

FIGURE 4.31 – Partie 9 - Métrique de la corruption de paquets

Les données récoltées lors de l'exécution de cette partie du script nous démontrent qu'à première vue, en visualisant la figure 4.32, la distribution des temps d'exécution en mode Opérations (O) semble relativement normale tandis que celle en mode Sabotages (S) ne semble pas l'être.

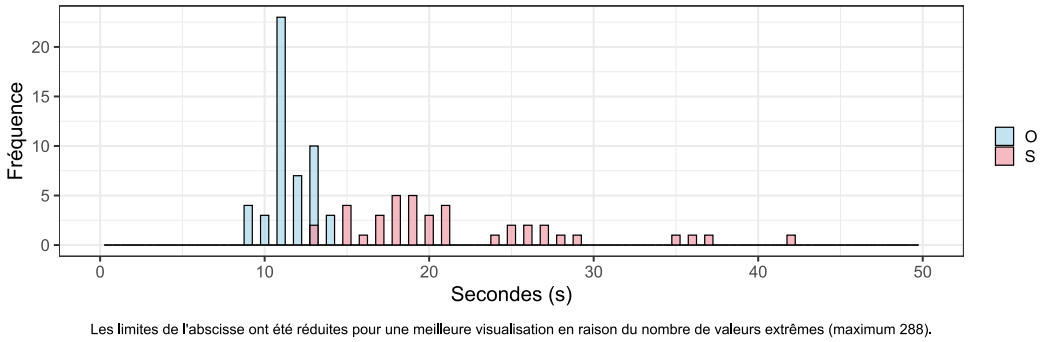

FIGURE 4.32 – Partie 9 - Fréquence de la durée de traitement

Le résultat du test Shapiro-Wilk sur chacune des distributions nous permet de constater qu'en fait, les deux ne le sont pas. Il est de 0.000431 pour le mode Opérations et de 6.646e-11 pour le mode Sabotages.

Comme les deux distributions ne sont pas normales, nous pouvons effectuer le test U de Mann–Whitney (non paramétrique) pour vérifier les deux groupes de données. Avec un résultat de 1.06e-17, tel que vu à la figure 4.33, la différence avec respect de la variable durée est statistiquement significative, nous pouvons donc conclure que ce sabotage a eu un certain effet sur la performance d'Ozone.

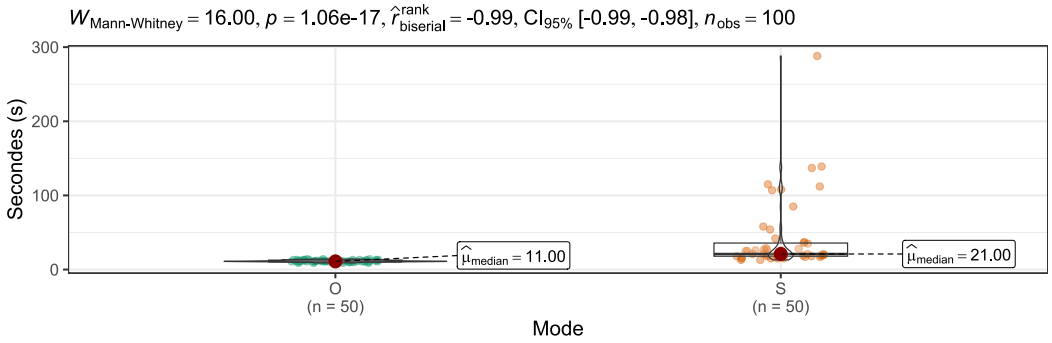

FIGURE 4.33 – Partie 9 - Comparaison de la durée de traitement

4.3.10 Partie 10 - Redémarrage immédiat d'une *node*

Pour ce dixième sabotage, aucune métrique (variable du script) ne venait altérer les effets sur la *node* affectée. Elle redémarrait sur le champ.

Les données récoltées lors de l'exécution de cette partie du script nous démontrent qu'à première vue, en visualisant la figure 4.34, la distribution des temps d'exécution en mode Opérations (O) ne semble pas normale tandis que celle en mode Sabotages (S) semble l'être.

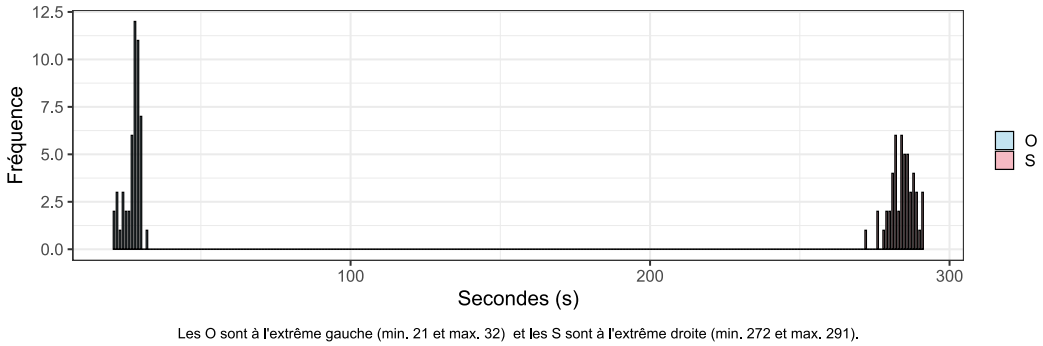

FIGURE 4.34 – Partie 10 - Fréquence de la durée de traitement

Le résultat du test Shapiro-Wilk sur chacune des distributions nous permet de constater que celle du mode Opérations n'est pas normale avec une valeur de 0.00014 contrairement à 0.28500 pour le mode Sabotages, indiquant que cette dernière l'est.

Comme au moins une des deux distributions n'est pas normale, nous pouvons effectuer le test U de Mann–Whitney (non paramétrique) pour vérifier les deux groupes de données. Avec un résultat de 5.96e-18, tel que vu à la figure 4.35, la différence avec respect de la variable durée est statistiquement significative, nous pouvons donc conclure que ce sabotage a eu un certain effet sur la performance d'Ozone.

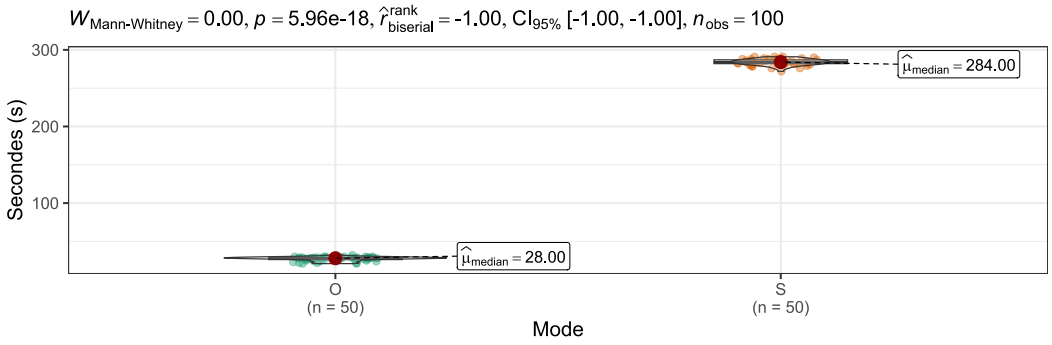

FIGURE 4.35 – Partie 10 - Comparaison de la durée de traitement

4.3.11 Constats

Nous pouvons constater dans le tableau 4.3, plus précisément les résultats obtenus au Test U, qu'il nous est possible d'assumer que nos sabotages ont eu un effet significatif sur la performance d'Ozone neuf (9) fois sur dix (10). Seul le Sabotage #1 a obtenu un résultat plus grand que 0.5, assumant ainsi que la surutilisation du processeur n'a pas causé de réel impact.

TABLEAU 4.3 – Durées moyennes selon le type d'exécution

P/S	μ O	μ S	% μ S	med(O)	med(S)	% med(S)	Test U
1	7 s	6.68 s	95.43	6 s	7 s	116.67	0.11080
2	32 s	40.32 s	126.00	37 s	41 s	110.81	2.633E-08
3	6 s	13.70 s	228.33	8 s	9 s	112.50	8.91E-05
4	53 s	220.74 s	416.49	48.50 s	235 s	484.54	6.84E-18
5	10 s	41.08 s	410.80	8 s	19 s	237.50	5.06E-18
6	42 s	114.96 s	273.71	36 s	96 s	266.67	1.88E-17
7	36 s	406.36 s	1128.78	40.50 s	110.50 s	272.84	9.76E-18
8	3 s	37.26 s	1242.00	4 s	7 s	175.00	3.43E-12
9	10 s	41.44 s	414.40	11 s	21 s	190.91	1.06E-17
10	29 s	284.08 s	979.59	28 s	284 s	1014.28	5.96E-18

En calculant le rang du résultat du Test U, nous pouvons aussi assumer que plus ce dernier est petit, plus le sabotage a eu un effet significatif. Selon le tableau 4.3, nous pouvons déduire que le délai réseau (S5), le redémarrage immédiat (S10) et l'arrêt d'une *node* (S4) forment le Top 3, suivi de très près par la réduction de la bande passante (S7).

En faisant l'exercice de comparer le pourcentage d'augmentation de la durée entre la moyenne et la médiane des modes Opérations et Sabotages, nous obtenons respectivement le redémarrage avec délai (S8), la réduction de bande passante (S7) et le redémarrage immédiat (S10) d'un côté et le redémarrage immédiat (S10), l'arrêt d'une *node* (S4) et la réduction de bande passante (S7) de l'autre. Le tableau 4.4 présente les rangs des sabotages les plus significatifs selon chaque méthode comparative que nous venons d'expliquer.

Il est important de mentionner qu'il y a probablement une incidence reliée à l'accumulation des sabotages sur les *nodes* de même que l'ordre de ces derniers et les opérations CRUD utilisées. De plus, le redémarrage d'une *node* à retardement (S8) exacerbe l'effet du redémarrage immédiat d'une *node* (S10) étant donné que ce sabotage risque de survenir au même moment que l'autre. Le nombre de *pipeline*s touchés peut donc être assez élevé.

Tableau 4.4 – Rangs des sabotages selon la durée d'exécution

Sabotage	Rang % μ S	Rang % med(S)	Rang Test U
S1	10	8	10
S2	9	10	8
S3	8	9	9
S4	4	2	3
S5	6	5	1
S6	7	4	6
S7	2	3	4
S8	1	7	7
S9	5	6	5
S10	3	1	2

Chapitre 5

CONCLUSION

Ce cinquième et dernier chapitre nous permet de faire le point quant à nos expérimentations et à l'atteinte de nos objectifs. Nous ferons d'abord un retour sur les résultats obtenus et l'hypothèse que nous avions posée. En deuxième lieu nous ferons un survol des contraintes et limites qui peuvent avoir influencé nos résultats pour finalement terminer en abordant quelques pistes et idées de travaux futurs et de perspectives de recherche en lien avec le système distribué Ozone.

5.1 Retour sur les résultats / hypothèse

Avec tous les systèmes existants dans notre société hautement digitalisée et dont la population n'a souvent aucune idée de l'infrastructure en place pour les soutenir, il est primordial et justifiable de mettre tout en oeuvre afin que ces derniers soient hautement disponibles. Il est encore plus impératif que leurs données soient fiables et cohérentes et ce, malgré toutes les failles qui peuvent survenir au quotidien (Maarten & Tanenbaum, 2016). C'est là l'essence même des systèmes distribués et des algorithmes de consensus tel que RAFT.

Nos résultats démontrent que malgré tous les sabotages exercés sur les *nodes*

de notre *cluster* Ozone, notre objectif principal a été atteint positivement. Ce dernier a pu demeurer fiable à 90% et cohérent à 100% tel que vu au tableau 4.2. Sa confiance ayant uniquement été ébranlée par nos défaillances en chaîne car sans elles, le système a obtenu une note parfaite. L'évaluation du temps d'exécution en mode Sabotages comparativement au mode Opérations nous permet d'assumer, outre la perte d'une *node*, que ce sont les sabotages qui affectent la communication et la transmission entre les *nodes* qui affectent le plus la performance d'Ozone.

Ces résultats confirment que l'algorithme de consensus RAFT via la librairie Ratis fonctionne très bien tout en étant en concordance avec le Théôrème CAP, sacrifiant ainsi la disponibilité (lenteur) au profit de la cohérence et la tolérance aux pannes et au partionnement réseau (Brewer, 2012). Le tableau 4.3 montre effectivement que les médianes de la durée de trait ement pour chacune des parties du scénario sont doublées sept fois sur dix en mode Sabotages par rapport au mode Opérations. De tels goulots d'étranglement surviennent pendant la réplication du *log* entre les *nodes* et demeure un sujet de recherche prisé (Jiang et al., 2023).

5.2 Limites

Nos travaux de recherche avaient bien évidemment leurs propres limites, c'est-à-dire une pluralité de contraintes qui peuvent avoir influencé les résultats obtenus :

- Documentation limitée ;
- Environnement de serveurs virtualisés (sept (7) clones identiques) ;
- Même réseau local et baie (*rack*) ;
- Mêmes ressources (hôtes physiques et stockage (*SAN*)) ;
- Aucune sécurité avancée ;
- Pour cette recherche, l'utilisation de HA pour OM et SCM n'a pas été

implémentée, bien que celle-ci puisse permettre un basculement rapide et transparent (*fail-over*) tout en assurant une plus grande résistance au niveau de la cohérence des métadonnées ;

— Lors de l'utilisation de Tc, les sabotages n'étaient pas cumulables pour une même *node*. Le sabotage en cours était réinitialisé avec la commande *tc qdisc del dev INTERFACE*.

En effet, notre *cluster* Ozone étant virtualisé, il partageait donc les mêmes ressources qu'environ deux cents (200) autres serveurs en fonction dans un environnement VMware, ce qui pouvait exercer une certaine influence. Pour cette même raison, il y avait peu d'intérêt à configurer une topologie *rack awareness* permettant à Ozone de sélectionner plus judicieusement les *nodes* d'un *pipeline* et mieux répartir les *replica*s des *container*s, tant avec RAFT que la réplication asynchrone du RM, ce qui aurait pu être bénéfique lors des simulations de défaillances.

Malgré tout, nous demeurons persuadés de la robustesse et de la validité des résultats obtenus selon les données recueillies dans les règles de l'art. Nous avions un échantillon de taille respectable avec cent (100) exécutions au total, divisé entre deux modes opératoires (Opérations et Sabotages), qui nous a permis de bien évaluer la fiabilité et la cohérence d'Ozone dans les deux situations, dites normales et extrêmes.

5.3 Travaux futurs

L'avenir s'avère plus que prometteur pour Ozone et plusieurs problématiques demeurent en suspens et à étudier. Nous avons remarqué dans nos expérimentations que la latence et la réduction de la bande passante d'une *node* peuvent grandement affecter le traitement de RAFT et créer une division réseau (Huang et al., 2018), il serait intéressant qu'une telle validation soit intégrée dans la politique

de sélection du *leader* (*ozone.scm.pipeline.leader-choose.policy*) ou même des *follower*s. Même si nous avons pu obtenir un rang et établir l'ordre d'impact des sabotages, il pourrait être intéressant de les tester un à la fois dans les mêmes conditions et en analyser l'impact sur les moyennes, les médianes et le Test de Mann-Whitney-Wilcoxon.

Aussi, bien que nous croyons que Ratis aurait dû être en mesure de prendre en charge le problème d'écriture lorsque l'espace disque est insuffisant sur une *node* d'un *pipeline*, l'utilisation du service *Container Balancer* d'Ozone aurait peut-être pu minimiser les impacts de notre sabotage #3 (*fallocate*).

De plus, la fiabilité et la cohérence des données avec RAFT serait-elle aussi efficace avec la réplication par *erasure coding* nouvellement ajoutée dans la version 1.3.0, laquelle pourrait réduire l'utilisation de stockage et du réseau (Wang et al., 2020) ?

Également, nous sommes conscients que la mise en place d'une infrastructure Ozone comprenant des serveurs géographiquement éloignés avec un ensemble de données de plus grande taille pourrait donner des résultats différents. De tels travaux seraient un excellent complément à notre sujet de recherche en plus d'adresser plusieurs problématiques de réseau plus difficiles à simuler notamment la scalabilité, la latence et le partitionnement.

Enfin, depuis la version 1.4.0, Ozone intègre une nouvelle méthode d'écriture afin d'éliminer les goulots d'étrangement (*bottlenecks*) engendrés par le *leader* qui a beaucoup plus de travail à effectuer que les *follower*s (Sze, 2022). Il serait assurément intéressant de s'y pencher et d'en comparer la performance.

Nous vous invitons à consulter notre dépôt GitHub, plateforme internationale de partage et contribution de code source, si vous désirez poursuivre nos travaux de recherche : https://github.com/nmathon7/MTI6500

RÉFÉRENCES

Agarwal, A. (2018). *Apache Hadoop Ozone - Object Store Architecture*. Cloudera Blog. Récupérée 28 décembre 2023, à partir de https://blog.cloudera.com/apache-hadoop-ozone-object-store-architecture/

Aggarwal, R., Verma, J., & Siwach, M. (2022). Small files' problem in Hadoop : A systematic literature review. *Journal of King Saud University - Computer and Information Sciences*, *34*(10, Part A), 8658-8674. https://doi.org/10.1016/j.jksuci.2021.09.007

Ahad, N., Sin Yin, T., Othman, A., & Yaacob, C. (2011). Sensitivity of Normality Tests to Non-normal Data. *Sains Malaysiana*, *40*, 637-641. http://journalarticle.ukm.my/2511/1/15_NorAishah.pdf

Ahn, J.-S., Kang, W.-H., Ren, K., Zhang, G., & Ben-Romdhane, S. (2019). Designing an Efficient Replicated Log Store with Consensus Protocol. *Proceedings of the 11th USENIX Conference on Hot Topics in Cloud Computing*, 8. https://dl.acm.org/doi/10.5555/3357034.3357044

Ahn, N., Black, J., & Effrat, J. (2001). *A Brief History of the Internet*. Distributed Computing. Récupérée 12 septembre 2022, à partir de https://cs.stanford.edu/people/eroberts/courses/soco/projects/2001-02/distributed-computing/home.html

Alain. (1985). *Propos sur le bonheur* (Gallimard, Éd.).

Ansari, J. W., Karim, N., Ansari, W., Beyan, O. D., & Cochez, M. (2018). *Semantic Profiling in Data Lake* (mém. de mast.). RWTH Aachen University, Germany. https://www.cochez.nl/teaching/supervision/theses/MScJasim.pdf

Bende, S., & Shedge, R. (2016). Dealing with Small Files Problem in Hadoop Distributed File System. *Procedia Computer Science*, *79*, 1001-1012. https://doi.org/10.1016/j.procs.2016.03.127

Brewer, E. A. (2012). CAP twelve years later : How the "rules" have changed. *Computer*, *45*, 23-29. https://doi.org/10.1109/MC.2012.37

Camacho-Rodríguez, J., Chauhan, A., Gates, A., Koifman, E., O'Malley, O., Garg, V., Haindrich, Z., Shelukhin, S., Jayachandran, P., Seth, S., Jaiswal, D., Bouguerra, S., Bangarwa, N., Hariappan, S., Agarwal, A., Dere, J., Dai, D., Nair, T., Dembla, N., ... Hagleitner, G. (2019). Apache Hive : From MapReduce to Enterprise-grade Big Data Warehousing. *Proceedings of the 2019 International Conference on Management of Data*, 1773-1786. https://doi.org/10.1145/3299869.3314045

Canonical. (2019). *Ubuntu Manpage Repository*. Ubuntu manuals. Récupérée 11 novembre 2023, à partir de https://manpages.ubuntu.com/

Cheng, L. (2020). *Multi-Raft – Boost up write performance for Apache Hadoop-Ozone*. Cloudera Blog. Récupérée 16 décembre 2023, à partir de https://blog.cloudera.com/multi-raft-boost-up-write-performance-for-apache-hadoop-ozone/

Chikh, F., & Warda, Z. (1996). *Mise en oeuvre d'un entrepôt de données sous Hadoop* (mém. de mast.). Université A/Mira de Béjaïa, Algérie.

Cloudera. (2021). *Creating a CRUD transactional table*. Cloudera Docs : Using Apache Hive. Récupérée 16 décembre 2023, à partir de https://docs.cloudera.com/cdw-runtime/cloud/using-hiveql/topics/hive_create_a_crud_transactional_table.html

Cloudera. (2023). *Storing data using Ozone*. Cloudera Docs : Using Apache Ozone. Récupérée 16 décembre 2023, à partir de https://docs.cloudera.com/cdp-private-cloud-base/latest/ozone-storing-data/ozone-storing-data.pdf

Dong, S., Callaghan, M., Galanis, L., Borthakur, D., Savor, T., & Strum, M. (2017). Optimizing Space Amplification in RocksDB. *Conference on Innovative Data Systems Research*, *3*, 3. https://www.eecg.toronto.edu/~stumm/Dong-CIDR-16.html

Elkawkagy, M., & Elbeh, H. (2020). High performance hadoop distributed file system. *International Journal of Networked and Distributed Computing*, *8*, 119-123. https://doi.org/10.2991/ijndc.k.200515.007

Excentis. (2014a). *Use Linux Traffic Control as impairment node in a test environment (part 1)*. Excentis Blog. Récupérée 13 novembre 2023, à partir de https://www.excentis.com/blog/use-linux-traffic-control-as-impairment-node-in-a-test-environment-part-1/

Excentis. (2014b). *Use Linux Traffic Control as impairment node in a test environment (part 2)*. Excentis Blog. Récupérée 13 novembre 2023, à partir de https://www.excentis.com/blog/use-linux-traffic-control-as-impairment-node-in-a-test-environment-part-2/

Fluri, C., Melnyk, D., & Wattenhofer, R. (2018). Improving Raft When There Are Failures. *2018 Eighth Latin-American Symposium on Dependable Computing (LADC)*, 167-170. https://doi.org/10.1109/LADC.2018.00028

Galery Käser, L. (2023). *Semi-Random Leader Election for Distributed Moving Target Defense Coordination in Kubernetes* (mém. de mast.). Umeå University, Sweden.

Hazelcast. (s. d.). *What is the CAP Theorem ?* Hazelcast. Récupérée 28 décembre 2023, à partir de https://hazelcast.com/glossary/cap-theorem/

Helvick, S. (2008). *A Survey of Hardware Performance Analysis Tools*. Récupérée 29 juin 2022, à partir de https://www.cse.wustl.edu/~jain/cse567-08/ftp/hw/index.html

Howard, H. (2014). *ARC : analysis of Raft consensus* (Computer Laboratory Tech. Rep. UCAM-CL-TR-857). University of Cambridge.

Howard, H., & Mortier, R. (2020). Paxos vs Raft : have we reached consensus on distributed consensus ? *Proceedings of the 7th Workshop on Principles and Practice of Consistency for Distributed Data*, 1-9. https://doi.org/10.1145/3380787.3393681

Howard, H., Schwarzkopf, M., Madhavapeddy, A., & Crowcroft, J. (2015). Raft Refloated : Do We Have Consensus ? *SIGOPS Oper. Syst. Rev.*, *49*(1), 12-21. https://doi.org/10.1145/2723872.2723876

Hu, J., & Liu, K. (2020). Raft consensus mechanism and the applications. *Journal of Physics : Conference Series*, *1544*(1), 012079. https://doi.org/10.1088/1742-6596/1544/1/012079

Huang, D., Ma, X., & Zhang, S. (2018). Performance Analysis of the Raft Consensus Algorithm for Private Blockchains. *IEEE Transactions on Systems, Man, and Cybernetics : Systems*, *50*, 172-181. https://doi.org/10.1109/TSMC.2019.2895471

Jain, E. P., & Gupta, E. A. (2017). Hadoop Architecture and Its Issues. *International Journal of Engineering Research and General Science*, *5*(2), 211-217.

Jain, R., Borkar, P., Deshmukh, P., Badhiye, S., Nimje, K., & Gupta, K. (2024). Choosing a Suitable Consensus Algorithm for Blockchain Applications : A Review of Factors and Challenges. *International Journal of Intelligent Systems and Applications in Engineering*, *12*, 333-341. https://ijisae.org/index.php/IJISAE/article/view/4381

Jankovic, S., Mladenovic, S., Mladenović, D., Veskovć, S., & Glavic, D. (2018). Schema on Read Modeling Approach as a Basis of Big Data

Analytics Integration with EIS. *Enterprise Information Systems*, *12*, 1180-1201. https://doi.org/10.1080/17517575.2018.1462404

Jiang, T., Huang, X., Song, S., Wang, C., Wang, J., Li, R., & Sun, J. (2023). Non-Blocking Raft for High Throughput IoT Data. *2023 IEEE 39th International Conference on Data Engineering (ICDE)*, 1140-1152. https://ieeexplore.ieee.org/document/10184820

Khan, R. Z. (2015). Distributed Computing : An Overview. *Int. J. Advanced Networking and Applications*, *07*, 2630-2635.

King, C. (2024). *GitHub repository - stress-ng*. Récupérée 4 mai 2024, à partir de https://github.com/ColinIanKing/stress-ng

Li, H., Liu, Z., & Li, Y. (2021). An improved raft consensus algorithm based on asynchronous batch processing. *Proceeding of 2021 International Conference on Wireless Communications, Networking and Applications*, 426-436. https://doi.org/10.1007/978-981-19-2456-9_44

Liu, X. (2018). *Weighted Raft and its Application to Geographically Distributed Servers* (mém. de mast.). University of Calgary, Canada.

Llave, M. R. (2018). Data lakes in business intelligence : reporting from the trenches. *Procedia Computer Science*, *138*, 516-524. https://doi.org/10.1016/j.procs.2018.10.071

Lu, S., Zhang, X., Zhao, R., Chen, L., Li, J., & Yang, G. (2023). P-Raft : An Efficient and Robust Consensus Mechanism for Consortium Blockchains. *Electronics*, *12*. https://doi.org/10.3390/electronics12102271

Maarten & Tanenbaum. (2016). A brief introduction to distributed systems. *Computing*, *98*(10), 967-1009. https://doi.org/10.1007/s00607-016-0508-7

Madera, C., & Laurent, A. (2016). The next information architecture evolution : the data lake wave. *Proceedings of the 8th International Conference on Management of Digital EcoSystems*, 174-180. https://doi.org/10.1145/3012071.3012077

Mehmood, H., Khalid, A., Kostakos, P., Gilman, E., & Pirttikangas, S. (2024). A novel Edge architecture and solution for detecting concept drift in smart environments. *Future Generation Computer Systems*, *150*, 127-143. https://doi.org/10.1016/j.future.2023.08.023

Mone, G. (2013). Beyond Hadoop. *Communications of the ACM*, *56*(1), 22. https://doi.org/10.1145/2398356.2398364

Nachar, N. (2008). The Mann-Whitney U : A Test for Assessing Whether Two Independent Samples Come from the Same Distribution. *Tutorials in Quantitative Methods for Psychology*, *4*. https://doi.org/10.20982/tqmp.04.1.p013

Nachiappan, R. (2020). *Efficient data reliability management of cloud storage systems for big data applications* (thèse de doct.). Western Sydney University, Australia.

Ongaro, D., & Ousterhout, J. (2014). In Search of an Understandable Consensus Algorithm. *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, 305-320. https://dl.acm.org/doi/10.5555/2643634.2643666

OpenWrt. (2018). *Netem (Network Emulator)*. OpenWrt. Récupérée 3 juillet 2023, à partir de https://openwrt.org/docs/guide-user/network/traffic-shaping/sch_netem

Ozone. (2023). *A high performance object store*. Apache Ozone. Récupérée 11 octobre 2022, à partir de https://ozone.apache.org/

Pareja Prieto, L. (2022). *Introducing Object Storage in Hadoop Ecosystem*. CERN. Récupérée 16 novembre 2023, à partir de https://cds.cern.ch/record/2835585

Pham, H. L., Tran, T. H., Duong Le, V. T., & Nakashima, Y. (2022). A High-Efficiency FPGA-Based Multimode SHA-2 Accelerator. *IEEE Access*, *10*, 11830-11845. https://doi.org/10.1109/ACCESS.2022.3146148

Quobyte. (s. d.). *What is HDFS, the Hadoop File System ?* Quobyte. Récupérée 2 février 2024, à partir de https://www.quobyte.com/storage-explained/what-is-hdfs/

Ratis. (2023). *Open source Java implementation for Raft consensus protocol*. Apache Ratis. Récupérée 11 octobre 2023, à partir de https://ratis.apache.org/

Sebaa, A., Chikh, F., Nouicer, A., & Tari, A. (2017). Research in Big Data Warehousing using Hadoop. *Journal of Information Systems Engineering & Management*, *2*(2), 10. https://doi.org/10.20897/jisem.201710

Shvachko, K., Kuang, H., Radia, S., & Chansler, R. (2010). The Hadoop Distributed File System. *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, 1-10. https://doi.org/10.1109/MSST.2010.5496972

Sze, T. (2019). *Apache Ratis - In Search of a Usable Raft Library [Fichier PowerPoint]*. Slideshare. Récupérée 28 octobre 2023, à partir de https://www.slideshare.net/TszWoNicholasSze/apache-ratis-in-search-of-a-usable-raft-library

Sze, T. (2022). *Ozone write pipeline V2 with Ratis streaming*. Cloudera Blog. Récupérée 16 décembre 2023, à partir de https://blog.cloudera.com/ozone-write-pipeline-v2-with-ratis-streaming/

Sztern, K. (2015). *Introduction aux systèmes distribués [Fichier vidéo]*. GConfs. Récupérée 7 mars 2023, à partir de https://www.youtube.com/watch?v=Q0QvyAUqnxI

Thusoo, A., Sarma, J. S., Jain, N., Shao, Z., Chakka, P., Zhang, N., Anthony, S., Liu, H., & Murthy, R. (2010). Hive - a petabyte scale data warehouse using Hadoop. *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, 996-1005. https://doi.org/10.1109/ICDE.2010.5447738

Université de Berkeley. (2023). *Goals*. Science United. Récupérée 11 octobre 2023, à partir de https://scienceunited.org/su_about.php

Vadivelu, N., Viswanadham, B., & Banerjee, S. (2020). *One billion files in Ozone*. Cloudera Blog. Récupérée 21 avril 2024, à partir de https://blog.cloudera.com/one-billion-files-in-ozone/

Vanlightly, J. (2019). *Quorum Queues Internals - A Deep Dive*. CloudAMQP. Récupérée 12 septembre 2022, à partir de https://www.cloudamqp.com/blog/quorum-queues-internals-a-deep-dive.html

VMware. (2024). *vSphere*. VMware. Récupérée 4 mai 2024, à partir de https://www.vmware.com/products/vsphere.html

Wang, Z. J., Li, T., Wang, H., Shao, A., Bai, Y., Cai, S., Xu, Z.-H., & Wang, D. (2020). CRaft : An Erasure-coding-supported Version of Raft for Reducing Storage Cost and Network Cost. *Proceedings of the 18th USENIX Conference on File and Storage Technologies*, 297-308. https://dl.acm.org/doi/10.5555/3386691.3386720

Woos, D., Wilcox, J. R., Anton, S., Tatlock, Z., Ernst, M. D., & Anderson, T. (2016). Planning for change in a formal verification of the raft consensus protocol. *Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs*, 154-165. https://doi.org/10.1145/2854065.2854081

Xia, Yun-Hao, Hong, Han-Shu, Lin, Guo-Feng & Sun, Zhi-Xin. (2017). Efficient Data Integrity Verification Using CRC Based on HDFS in Cloud Storage. *ITM Web of Conferences*, *11*, 10002. https://doi.org/10.1051/itmconf/20171110002

Zhang, Y., Han, B., Zhang, Z.-L., & Gopalakrishnan, V. (2017). Network-Assisted Raft Consensus Algorithm. *Proceedings of the SIGCOMM Posters and Demos*, 94-96. https://doi.org/10.1145/3123878.3131998

Annexe A

DONNÉES BRUTES

Cette Annexe vous présente les données brutes recueillies lors des nos expérimentations avec Ozone.

Dans le tableau A.1, nous présentons les résultats des journaux de chacune des exécutions. Étant donné le grand nombre de colonnes, les entêtes ont été raccourcies. Voici la description de chacune d'elles :

- Sé : Séquence ;
- Size : Taille totale des données du *cluster* ;
- Cont : Nombre de *containers* ;
- TU[1-7] : Capacité utilisée (*Total Used*) pour les *nodes* 1 à 7 ;
- OU[1-7] : Capacité utilisée par Ozone (*Ozone Used*) pour les *nodes* 1 à 7 ;
- TR[1-7] : Capacité restante (*Total Remaining*) pour les *nodes* 1 à 7 ;
- R : Résultat (S pour Succès, E pour Échec).

Dans le tableau A.2, nous présentons la durée totale de chacune des exécutions.

Dans le tableau A.3, nous présentons la durée de chacune des parties (opérations CRUD) des exécutions.

Dans le tableau A.4, nous présentons la compilation des sabotages.

Dans le tableau A.5, nous présentons la compilation des sabotages affectés à chaque *node* du *cluster* pendant les exécutions en mode Sabotages.

Dans le tableau A.6, nous présentons les valeurs des métriques (variables) utilisées pendant les exécutions en mode Sabotages.

Tableau A.1 – Résultats des exécutions

Sé	Size	Cont	TU1	OU1	TR1	TU2	OU2	TR2	TU3	OU3	TR3	TU4	OU4	TR4	TU5	OU5	TR5	TU6	OU6	TR6	TU7	OU7	TR7	R
0	2.30	5	12.23	0.72	87.77	1.12	0.72	98.88	2.04	1.64	97.96	3.24	2.84	96.76	2.40	2.00	97.60	2.40	2.40	97.60	1.39	1.00	98.61	S
1	3.50	13	12.42	2.43	87.58	2.87	0.98	97.13	3.60	1.71	96.40	5.83	5.47	94.17	5.07	4.72	94.93	5.07	4.72	94.93	3.22	2.87	96.78	S
2	10.10	15	12.80	2.75	87.20	2.61	2.26	97.39	3.94	3.54	96.06	7.08	6.72	92.92	4.71	4.36	95.29	6.40	6.05	93.60	2.02	1.67	97.98	S
3	13.20	21	12.13	2.73	87.87	2.70	2.31	97.30	4.00	3.61	96.00	7.16	6.78	92.84	7.38	7.00	92.62	7.39	7.01	92.61	4.41	4.03	95.59	S
4	10.40	35	13.34	3.91	86.66	1.22	0.83	98.78	5.18	4.79	94.82	7.35	6.96	92.65	7.68	7.29	92.32	7.69	7.30	92.31	4.23	3.84	95.77	S
5	12.50	39	14.68	5.20	85.32	1.22	0.83	98.78	6.48	6.09	93.52	7.64	7.25	92.36	7.83	7.44	92.17	7.84	7.45	92.16	6.87	6.48	93.13	S
6	11.40	41	16.83	7.33	83.17	2.13	1.73	97.87	7.43	7.04	92.57	4.77	4.38	95.23	6.82	6.44	93.18	3.99	3.60	96.01	7.29	6.90	92.71	S
7	5.20	24	18.95	8.63	81.05	3.89	3.47	96.11	10.25	9.83	89.75	5.16	4.74	94.84	7.10	6.68	92.90	4.07	3.65	95.93	8.50	8.08	91.50	S
8	6.10	29	18.89	8.54	81.11	8.89	8.46	91.11	10.40	9.98	89.60	5.46	5.04	94.54	9.17	8.75	90.83	3.99	3.57	96.01	5.07	4.64	94.93	S
9	6.10	33	19.27	8.87	80.73	10.00	9.58	90.00	10.78	10.35	89.22	6.57	6.15	93.43	10.24	9.82	89.76	4.33	3.91	95.67	5.07	4.65	94.93	S
10	6.60	39	19.97	9.55	80.03	11.17	10.75	88.83	11.69	11.26	88.31	8.43	8.01	91.57	11.31	10.89	88.69	3.93	3.50	96.07	3.65	3.23	96.35	S
11	7.00	43	22.55	9.96	77.45	12.60	12.09	87.40	13.06	12.56	86.94	9.86	9.35	90.14	11.85	11.35	88.15	4.16	3.66	95.84	5.63	5.12	94.37	S
12	8.10	48	25.74	14.12	74.26	14.78	12.36	85.22	12.80	12.80	87.20	8.80	8.29	91.20	13.95	13.45	86.05	4.09	3.66	95.91	7.14	6.63	92.86	S
13	8.40	50	26.46	13.55	73.54	15.88	15.36	84.12	11.63	11.11	88.37	9.32	8.80	90.68	14.27	13.75	85.73	5.28	4.76	94.72	8.13	7.61	91.87	S
14	9.20	54	27.61	14.68	72.39	16.52	16.00	83.48	11.83	11.31	88.17	2.69	2.17	97.31	15.39	14.87	84.61	4.53	4.01	95.47	13.79	13.26	86.21	S
15	9.20	56	28.91	15.95	71.09	16.70	16.18	83.30	12.03	11.50	87.97	2.69	2.17	97.31	16.66	16.14	83.34	6.20	5.67	93.80	16.52	16.00	83.48	S
16	9.50	62	26.96	13.98	73.04	15.65	15.13	84.35	12.29	11.77	87.71	1.27	0.75	98.73	20.89	20.37	79.11	6.42	5.89	93.58	17.72	17.19	82.28	S
17	10.10	66	28.78	15.18	71.22	15.72	15.17	84.28	13.59	13.05	86.41	2.76	2.22	97.24	22.79	22.25	77.21	6.21	5.67	93.79	19.68	19.13	80.32	S
18	6.20	70	23.42	8.28	76.58	7.22	6.67	92.78	7.19	6.64	92.81	2.43	1.88	97.57	13.19	12.64	86.81	2.88	2.33	97.12	10.18	9.62	89.82	S
19	6.60	74	26.73	9.42	73.27	7.91	7.28	92.09	8.54	7.91	91.46	2.66	2.03	97.34	13.79	13.17	86.21	4.18	3.55	95.82	10.73	10.10	89.27	S
20	7.30	79	27.24	9.69	72.76	8.91	8.91	91.09	10.11	9.28	89.89	3.01	2.03	96.99	15.45	14.43	84.55	3.75	3.61	96.25	11.27	10.63	88.73	S
21	7.80	85	28.41	11.08	71.59	10.75	10.11	89.25	11.13	10.30	88.87	3.29	2.31	96.71	15.68	14.66	84.32	3.76	3.61	96.24	11.69	11.06	88.31	S
22	7.80	89	23.06	6.73	76.94	12.33	11.70	87.67	11.85	11.85	88.15	4.69	4.06	95.31	14.91	14.91	85.09	5.72	5.72	94.28	11.66	8.21	88.34	S
23	8.20	89	23.44	6.08	76.56	12.67	12.04	87.33	11.01	10.38	88.99	5.77	5.14	94.23	10.12	9.49	89.88	5.33	4.69	94.67	12.74	12.10	87.26	S
24	8.20	91	24.94	7.57	75.06	12.25	12.25	87.75	11.68	11.05	88.32	7.29	6.66	92.71	10.80	9.49	89.20	6.83	5.83	93.17	14.88	14.25	85.12	S
25	8.40	97	25.28	7.91	74.72	10.01	9.37	89.99	11.94	11.30	88.06	7.55	6.92	92.45	11.99	11.36	88.01	8.02	7.38	91.98	15.99	15.35	84.01	S
26	9.00	99	26.71	9.31	73.29	12.82	12.18	87.18	10.44	9.81	89.56	6.93	6.30	93.07	13.83	13.20	86.17	9.01	8.38	90.99	16.32	15.69	83.68	S
27	9.30	101	27.41	9.55	72.59	13.97	13.31	86.03	11.50	10.85	88.50	8.09	7.43	91.91	14.23	13.58	85.77	9.41	8.76	90.59	16.61	15.96	83.39	S
28	10.10	103	27.92	12.16	72.08	16.56	15.9	83.44	10.85	10.85	89.15	9.72	9.72	90.28	13.89	13.84	86.11	9.02	9.02	90.98	20.95	20.29	79.05	S
29	10.60	103	29.17	13.40	70.83	16.85	16.18	83.15	10.85	10.85	89.15	10.01	10.01	89.99	16.53	15.00	83.47	9.84	9.19	90.16	21.15	20.49	78.85	S
30	11.10	105	32.25	14.35	67.75	17.53	16.87	82.47	8.56	7.90	91.44	10.51	10.51	89.49	17.05	16.40	82.95	9.37	9.37	90.63	25.20	24.54	74.80	S
31	11.20	107	33.41	15.50	66.59	18.50	17.84	81.50	9.03	8.37	90.97	11.44	10.79	88.56	17.34	16.68	82.66	5.93	5.28	94.07	25.39	24.73	74.61	S
32	11.70	111	33.40	15.48	66.60	20.09	19.44	79.91	13.84	13.19	86.16	10.74	10.74	89.26	19.26	18.61	80.74	5.30	5.30	94.70	25.04	25.04	74.96	S
33	12.20	113	33.67	15.74	66.33	20.36	19.69	79.64	15.02	14.37	84.98	12.10	12.10	87.90	20.44	19.79	79.56	5.04	4.38	94.96	24.26	23.60	75.74	S
34	13.20	117	22.78	3.71	77.22	9.16	8.38	90.84	6.19	5.41	93.81	2.13	1.36	97.87	8.49	7.72	91.51	2.93	2.14	97.07	13.00	12.22	87.00	S
35	6.80	125	23.10	4.03	76.90	9.64	8.86	90.36	8.64	7.86	91.36	2.13	1.36	97.87	10.95	10.17	89.05	5.54	4.76	94.46	13.48	12.70	86.52	S
36	7.40	127	23.43	4.35	76.57	9.17	9.00	90.83	10.49	9.71	89.51	3.62	1.36	96.38	12.21	11.32	87.79	7.30	6.41	92.70	14.11	13.33	85.89	S
37	7.80	131	23.73	4.65	76.27	10.11	9.95	89.89	10.77	9.99	89.23	4.78	4.01	95.22	12.49	11.71	87.51	7.72	6.93	92.28	15.06	14.28	84.94	S
38	8.50	135	23.91	4.81	76.09	11.35	11.35	88.65	11.07	10.29	88.93	4.21	4.21	95.79	15.07	12.01	84.93	8.96	8.16	91.04	16.76	15.98	83.24	S
39	8.90	135	25.50	6.09	74.50	11.75	10.96	88.25	12.44	11.65	87.56	3.13	2.34	96.87	16.44	15.65	83.56	9.18	8.38	90.82	16.89	16.10	83.11	S
40	9.60	140	27.42	7.28	72.58	13.49	12.06	86.51	14.44	11.99	85.56	5.11	2.38	94.89	15.94	15.94	84.06	8.59	8.59	91.41	18.22	17.25	81.78	S
41	10.00	142	28.58	9.15	71.42	13.57	12.77	86.43	15.81	15.02	84.19	5.42	4.63	94.58	17.94	17.15	82.06	4.26	3.46	95.74	18.45	17.66	81.55	S

Suite sur la prochaine page...

Tableau A.1 – Suite de la page précédente

Séq	Size	Cont	TU1	OU1	TR1	TU2	OU2	TR2	TU3	OU3	TR3	TU4	OU4	TR4	TU5	OU5	TR5	TU6	OU6	TR6	TU7	OU7	TR7	R
42	10.50	147	30.26	10.73	69.74	20.41	19.61	79.59	11.49	10.69	88.51	4.70	3.90	95.30	20.21	19.42	79.79	3.42	2.62	96.58	18.64	17.85	81.36	S
43	10.50	149	31.31	11.78	68.69	21.60	20.80	78.40	11.49	10.69	88.51	5.09	4.30	94.91	20.69	19.89	79.31	3.67	2.87	96.33	19.61	18.82	80.39	S
44	10.90	153	32.74	13.02	67.26	23.22	22.41	76.78	13.44	12.64	86.56	3.16	2.36	96.84	19.96	19.16	80.04	2.85	2.04	97.15	20.79	19.99	79.21	S
45	11.40	157	33.04	13.32	66.96	23.56	22.75	76.44	14.59	13.79	85.41	3.16	2.36	96.84	21.58	19.30	78.42	5.44	3.14	94.56	22.09	21.29	77.91	S
46	11.60	157	34.10	14.37	65.90	23.78	22.97	76.22	14.18	14.18	85.82	2.36	2.36	97.64	22.66	21.86	77.34	5.85	5.85	94.15	22.46	21.66	77.54	S
47	12.20	157	35.86	15.63	64.14	24.89	24.06	75.11	15.27	14.44	84.73	1.74	0.91	98.26	23.08	22.25	76.92	7.12	6.29	92.88	23.61	22.79	76.39	S
48	12.40	159	36.17	15.93	63.83	26.08	25.25	73.92	15.59	15.59	84.41	1.74	0.91	98.26	23.44	23.44	76.56	7.37	6.54	92.63	23.90	23.08	76.10	S
49	13.10	163	37.50	17.26	62.50	27.31	26.48	72.69	16.49	15.66	83.51	3.30	0.99	96.70	24.48	23.65	75.52	7.59	6.76	92.41	25.10	24.27	74.90	S
50	9.10	168	26.61	6.34	73.39	11.72	10.89	88.28	7.14	6.80	92.86	3.37	0.36	96.63	14.13	13.36	85.87	7.11	7.11	92.89	22.82	22.82	77.18	S
51	8.90	172	24.82	6.72	75.18	11.62	10.78	88.38	8.21	7.37	91.79	4.30	3.46	95.70	15.31	14.47	84.69	6.74	5.90	93.26	22.62	21.79	77.38	S
52	9.00	174	26.02	7.90	73.98	11.74	10.89	88.26	8.88	8.04	91.12	3.63	3.63	96.37	15.94	15.94	84.06	7.92	7.08	92.08	22.69	21.86	77.31	S
53	9.60	176	27.24	9.12	72.76	11.96	11.12	88.04	9.17	8.33	90.83	4.91	3.75	95.09	17.93	17.07	82.07	9.14	8.30	90.86	22.82	21.98	77.18	S
54	9.50	182	24.95	6.71	75.05	15.81	14.96	84.19	10.29	9.45	89.71	5.45	4.61	94.55	15.69	14.85	84.31	3.56	2.71	96.44	21.47	20.64	78.53	S
55	9.50	188	26.54	7.84	73.46	17.09	16.23	82.91	10.76	9.90	89.24	5.70	4.84	94.30	17.58	16.72	82.42	4.63	3.77	95.37	21.52	20.67	78.48	S
56	10.00	188	27.56	8.88	72.44	18.23	17.36	81.77	11.06	10.20	88.94	6.01	5.15	93.99	17.97	17.11	82.03	5.78	4.91	94.22	21.53	20.67	78.47	S
57	10.60	192	28.03	9.26	71.97	19.15	18.29	80.85	12.27	11.41	87.73	6.42	5.56	93.58	19.01	18.15	80.99	6.02	5.16	93.98	23.14	22.28	76.86	S
58	11.10	196	29.60	10.83	70.40	19.74	18.50	80.26	11.68	11.68	88.32	5.69	5.69	94.31	19.56	18.39	80.44	8.13	6.32	91.87	25.19	23.40	74.81	S
59	11.60	198	30.83	12.01	69.17	20.01	19.14	79.99	13.17	12.30	86.83	3.34	2.47	96.66	19.85	18.99	80.15	9.28	8.41	90.72	26.34	25.48	73.66	S
60	12.60	202	26.07	6.88	73.93	14.56	13.67	85.44	7.11	6.22	92.89	3.81	2.93	96.19	14.02	13.14	85.98	4.54	3.65	95.46	23.59	22.70	76.41	S
61	9.10	208	27.28	7.81	72.72	15.62	14.72	84.38	7.54	6.64	92.46	4.25	3.35	95.75	14.29	13.40	85.71	4.98	4.08	95.02	24.53	23.63	75.47	S
62	9.50	219	28.39	8.82	71.61	16.91	15.97	83.09	8.34	7.29	91.66	3.51	3.51	96.49	14.83	13.94	85.17	4.60	4.60	95.40	26.23	24.67	73.77	S
63	10.00	221	29.58	10.00	70.42	17.20	16.30	82.80	9.43	8.53	90.57	3.68	3.68	96.32	15.09	14.20	84.91	5.68	4.77	94.32	27.41	26.51	72.59	S
64	10.50	227	29.65	12.38	70.35	18.85	17.48	81.15	9.93	9.01	90.07	4.30	3.40	95.70	14.48	14.44	85.52	5.57	4.92	94.43	29.65	12.38	70.35	S
65	10.90	231	30.89	13.61	69.11	19.13	18.23	80.87	11.10	10.20	88.90	4.51	3.61	95.49	14.48	14.44	85.52	5.78	5.13	94.22	32.03	31.13	67.97	S
66	11.50	238	32.69	14.49	67.31	19.66	19.44	80.34	10.36	10.36	89.64	5.30	5.16	94.70	16.90	16.01	83.10	6.67	6.60	93.33	32.16	31.40	67.84	S
67	12.10	244	33.02	14.81	66.98	21.44	19.73	78.56	11.32	11.32	88.68	5.64	5.50	94.36	18.05	17.15	81.95	6.97	6.89	93.03	34.61	32.36	65.39	S
68	12.50	244	31.94	14.79	68.06	24.82	23.92	75.18	8.47	8.47	91.53	10.76	10.76	89.24	18.22	17.43	81.78	8.03	7.26	91.97	32.45	32.45	67.55	S
69	13.00	246	34.02	13.56	65.98	25.95	25.02	74.05	8.03	7.10	91.97	6.24	5.31	93.76	20.20	19.27	79.80	9.16	8.23	90.84	31.75	30.82	68.25	S
70	12.90	251	34.75	14.27	65.25	26.68	25.74	73.32	9.36	8.43	90.64	5.63	5.63	94.37	21.16	20.36	78.84	9.54	8.60	90.46	31.83	31.8	68.17	S
71	13.50	255	35.03	14.54	64.97	27.91	26.97	72.09	10.57	9.64	89.43	5.72	5.72	94.28	23.82	21.54	76.18	9.81	8.87	90.19	33.40	31.89	66.60	S
72	14.00	255	34.68	14.18	65.32	30.64	28.20	69.36	10.98	10.98	89.02	5.77	5.77	94.23	26.10	25.17	73.90	9.50	9.08	90.50	33.31	31.93	66.69	S
73	14.40	257	35.11	14.59	64.89	31.05	30.12	68.95	10.93	9.99	89.07	4.55	3.61	95.45	27.14	26.21	72.86	9.91	8.97	90.09	34.19	33.25	65.81	S
74	10.90	261	29.25	7.85	70.75	26.88	26.88	73.12	5.72	3.50	94.28	5.92	3.19	94.08	16.36	14.59	83.64	5.64	4.75	94.36	28.40	26.50	71.60	S
75	11.30	263	29.37	7.17	70.63	24.69	23.75	75.31	6.97	6.03	93.03	5.78	4.84	94.22	17.54	16.60	82.46	5.96	5.01	94.04	29.57	28.63	70.43	S
76	11.30	268	30.40	8.38	69.60	25.11	23.78	74.89	8.06	7.22	91.94	6.28	4.17	93.72	15.88	15.88	84.12	7.18	6.22	92.82	29.14	27.94	70.86	S
77	11.70	270	30.88	8.65	69.12	26.10	25.15	73.90	9.23	8.28	90.77	6.55	5.61	93.45	15.15	14.21	84.85	7.64	6.68	92.36	30.13	29.18	69.87	S
78	12.20	272	31.42	8.93	68.58	26.15	25.20	73.85	10.63	9.67	89.37	5.22	4.26	94.78	18.22	17.27	81.78	8.67	7.70	91.33	29.18	28.22	70.82	S
79	12.40	274	28.69	8.14	71.31	22.66	21.70	77.34	6.67	5.72	93.33	4.71	3.75	95.29	15.56	14.61	84.44	8.17	7.21	91.83	26.39	25.44	73.61	S
80	10.80	282	29.14	8.58	70.86	24.74	21.75	75.26	6.93	6.93	93.07	3.93	3.93	96.07	17.03	16.00	82.97	9.39	8.43	90.61	25.95	25.95	74.05	S
81	11.50	284	30.34	9.77	69.66	24.77	23.81	75.23	9.69	8.35	90.31	3.95	3.95	96.05	17.26	16.31	82.74	10.81	9.84	89.19	26.94	25.97	73.06	S
82	12.20	288	31.30	14.69	68.70	24.70	22.57	75.30	11.38	11.38	88.62	7.69	7.69	92.31	18.00	16.55	82.00	10.93	10.93	89.07	28.32	27.36	71.68	S
83	12.30	288	31.76	10.92	68.24	24.72	23.75	75.28	10.99	10.02	89.01	5.20	4.23	94.80	20.78	19.81	79.22	10.49	9.51	89.51	29.55	28.58	70.45	S
84	13.50	293	31.92	11.04	68.08	25.58	24.61	74.42	11.11	10.14	88.89	6.57	5.60	93.43	19.02	18.05	80.98	10.63	9.65	89.37	30.57	29.60	69.43	S
85	13.10	299	33.02	12.14	66.98	26.60	25.63	73.40	11.32	10.35	88.68	6.84	5.87	93.16	19.30	18.33	80.70	10.84	9.86	89.16	33.29	32.32	66.71	S

Suite sur la prochaine page...

Tableau A.1 – Suite de la page précédente

Séq	Size	Cont	TU1	OU1	TR1	TU2	OU2	TR2	TU3	OU3	TR3	TU4	OU4	TR4	TU5	OU5	TR5	TU6	OU6	TR6	TU7	OU7	TR7	R
86	13.20	302	32.25	11.12	67.75	30.25	29.27	69.75	9.86	8.88	90.14	3.68	2.70	96.32	20.12	19.14	79.88	9.35	8.36	90.65	34.59	33.62	65.41	S
87	13.50	305	33.44	12.27	66.56	31.43	30.45	68.57	11.48	10.50	88.52	3.82	2.84	96.18	20.43	19.46	79.57	9.66	8.68	90.34	35.73	34.75	64.27	S
88	14.20	310	34.18	12.98	65.82	28.46	27.48	71.54	15.22	14.24	84.78	4.35	3.36	95.65	20.33	19.35	79.67	10.71	9.72	89.29	38.07	37.09	61.93	S
89	14.40	314	34.64	13.18	65.36	29.61	28.62	70.39	15.45	14.46	84.55	4.57	3.58	95.43	21.59	20.60	78.41	11.86	10.86	88.14	38.32	37.33	61.68	S
90	15.40	314	35.73	14.26	64.27	31.27	30.27	68.73	15.97	14.82	84.03	4.78	3.94	95.22	23.54	22.56	76.46	10.95	10.95	89.05	43.79	42.80	56.21	S
91	15.80	316	36.10	14.63	63.90	31.36	30.37	68.64	17.19	16.04	82.81	6.00	5.16	94.00	23.77	22.77	76.23	11.05	11.05	88.95	44.89	43.9	55.11	S
92	16.20	320	33.60	12.12	66.40	32.35	31.34	67.65	16.30	16.19	83.70	6.18	5.19	93.82	28.65	24.04	71.35	12.01	12.01	87.99	45.82	44.83	54.18	S
93	16.60	320	33.87	12.39	66.13	33.53	32.53	66.47	16.30	15.30	83.70	6.54	5.55	93.46	29.74	28.75	70.26	7.60	6.60	92.40	46.08	45.09	53.92	S
94	19.20	326	25.44	9.07	74.56	26.34	21.99	73.66	7.01	7.01	92.99	8.20	8.20	91.80	18.39	17.40	81.61	5.72	5.72	94.28	44.27	43.26	55.73	S
95	12.60	328	26.57	10.19	73.43	26.55	25.55	73.45	6.97	5.97	93.03	6.03	5.04	93.97	18.71	17.71	81.29	5.02	4.01	94.98	44.47	43.47	55.53	S
96	12.80	328	26.81	4.68	73.19	26.43	25.40	73.57	9.52	8.50	90.48	7.34	6.32	92.66	20.13	19.11	79.87	5.28	4.24	94.72	43.24	42.21	56.76	S
97	13.40	330	28.08	5.95	71.92	26.83	25.80	73.17	9.70	8.67	90.30	8.34	7.32	91.66	20.31	19.29	79.69	6.19	5.16	93.81	45.12	44.09	54.88	S
98	13.80	330	29.14	7.09	70.86	27.18	25.91	72.82	9.93	8.97	90.07	9.55	8.52	90.45	21.00	19.59	79.00	7.47	6.31	92.53	44.73	44.21	55.27	S
99	14.20	332	30.14	7.98	69.86	28.12	27.10	71.88	10.39	9.36	89.61	9.78	8.75	90.22	21.47	20.44	78.53	7.79	6.76	92.21	45.66	44.64	54.34	S
100	15.00	337	33.24	11.06	66.76	29.90	28.12	70.10	10.64	9.62	89.36	8.99	8.99	91.01	21.85	20.73	78.15	9.73	7.41	90.27	48.00	45.63	52.00	S

Tableau A.2 – Durée totale de chacune des exécutions

Séquence	Mode	Durée	Secondes (s)	Séquence	Mode	Durée	Secondes (s)
1	O	8 min 13 s	493	51	O	7 min 51 s	471
2	S	28 min 29 s	1709	52	S	24 min 17 s	1457
3	O	8 min 15 s	495	53	O	7 min 23 s	443
4	S	35 min 4 s	2104	54	S	19 min 31 s	1171
5	O	8 min 29 s	509	55	O	8 min 24 s	504
6	S	22 min 8 s	1328	56	S	14 min 27 s	867
7	O	8 min 50 s	530	57	O	7 min 29 s	449
8	S	24 min 31 s	1471	58	S	24 min 45 s	1485
9	O	8 min 11 s	491	59	O	7 min 53 s	473
10	S	21 min 59 s	1319	60	S	21 min 15 s	1275
11	O	9 min 7 s	547	61	O	7 min 59 s	479
12	S	19 min 6 s	1146	62	S	20 min 52 s	1252
13	O	8 min 33 s	513	63	O	7 min 40 s	460
14	S	18 min 3 s	1083	64	S	18 min 54 s	1134
15	O	8 min 6 s	486	65	O	7 min 46 s	466
16	S	25 min 0 s	1500	66	S	38 min 47 s	2327
17	O	8 min 28 s	508	67	O	7 min 48 s	468
18	S	19 min 42 s	1182	68	S	24 min 29 s	1469
19	O	9 min 13 s	553	69	O	8 min 1 s	481
20	S	24 min 40 s	1480	70	S	38 min 59 s	2339
21	O	8 min 22 s	502	71	O	7 min 7 s	427
22	S	19 min 40 s	1180	72	S	23 min 17 s	1397
23	O	8 min 48 s	528	73	O	7 min 42 s	462
24	S	17 min 47 s	1067	74	S	17 min 45 s	1065
25	O	8 min 49 s	529	75	O	8 min 6 s	486
26	S	21 min 47 s	1307	76	S	20 min 48 s	1248
27	O	8 min 23 s	503	77	O	7 min 45 s	465
28	S	32 min 30 s	1950	78	S	143 min 38 s	8618
29	O	8 min 7 s	487	79	O	7 min 10 s	430
30	S	20 min 29 s	1229	80	S	21 min 37 s	1297
31	O	8 min 41 s	521	81	O	8 min 6 s	486
32	S	17 min 43 s	1063	82	S	17 min 36 s	1056
33	O	10 min 26 s	626	83	O	8 min 10 s	490
34	S	18 min 40 s	1120	84	S	20 min 42 s	1242
35	O	8 min 48 s	528	85	O	8 min 5 s	485
36	S	21 min 24 s	1284	86	S	45 min 30 s	2730
37	O	8 min 42 s	522	87	O	7 min 43 s	463
38	S	23 min 7 s	1387	88	S	18 min 19 s	1099
39	O	8 min 31 s	511	89	O	8 min 27 s	507
40	S	18 min 57 s	1137	90	S	17 min 12 s	1032
41	O	7 min 44 s	464	91	O	7 min 53 s	473
42	S	18 min 59 s	1139	92	S	16 min 42 s	1002
43	O	8 min 25 s	505	93	O	7 min 57 s	477
44	S	19 min 55 s	1195	94	S	33 min 40 s	2020
45	O	7 min 38 s	458	95	O	7 min 44 s	464
46	S	18 min 5 s	1085	96	S	17 min 7 s	1027
47	O	7 min 48 s	468	97	O	7 min 34 s	454
48	S	25 min 40 s	1540	98	S	18 min 55 s	1135
49	O	8 min 5 s	485	99	O	7 min 54 s	474
50	S	29 min 18 s	1758	100	S	18 min 0 s	1080

Tableau A.3 – Durée (s) des parties pour chacune des exécutions

Séquence	Mode	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
1	O	7	32	6	53	10	42	36	3	10	29
2	S	6	35	9	303	87	295	354	6	26	289
3	O	7	31	7	40	7	33	40	4	12	22
4	S	7	46	11	58	14	52	936	335	21	291
5	O	6	37	7	44	7	28	40	5	13	26
6	S	8	42	7	235	82	202	107	25	21	290
7	O	8	37	8	53	9	33	43	3	13	29
8	S	6	43	10	211	71	257	266	6	20	289
9	O	7	40	7	51	8	38	42	4	14	27
10	S	7	42	6	270	14	100	197	38	27	288
11	O	5	29	6	44	8	34	42	4	11	28
12	S	8	41	10	291	14	100	75	6	20	283
13	O	8	35	9	56	10	35	44	5	12	29
14	S	7	40	10	211	36	65	60	11	19	289
15	O	6	39	9	51	9	39	43	5	11	30
16	S	6	45	8	161	23	155	472	5	19	288
17	O	7	38	8	49	8	36	41	3	11	23
18	S	7	44	9	182	55	114	108	6	25	282
19	O	6	37	9	47	9	36	42	6	11	30
20	S	7	48	9	289	13	89	233	110	85	286
21	O	6	38	9	50	9	34	42	4	11	29
22	S	6	40	8	219	77	175	49	7	20	284
23	O	6	28	5	44	8	37	41	5	13	28
24	S	6	36	7	215	17	46	85	41	27	291
25	O	7	38	8	48	8	31	48	4	11	30
26	S	6	48	11	261	98	190	57	7	35	291
27	O	6	37	7	48	7	37	39	3	9	21
28	S	6	41	10	240	13	81	772	190	19	288
29	O	8	38	7	52	8	37	40	5	13	28
30	S	7	42	12	236	25	55	128	14	115	285
31	O	7	42	8	47	7	35	39	4	12	30
32	S	8	41	10	130	17	142	98	6	18	288
33	O	6	39	8	49	10	30	49	5	13	28
34	S	7	41	9	243	91	52	49	5	21	282
35	O	6	35	8	52	10	38	42	5	13	28
36	S	6	46	11	222	18	222	109	5	37	286
37	O	7	36	8	51	9	34	41	3	11	28
38	S	7	39	8	232	20	166	124	65	137	278
39	O	6	32	8	46	9	36	47	5	11	29
40	S	6	37	7	266	90	44	87	6	17	280
41	O	6	35	10	49	9	39	39	5	14	28
42	S	7	45	10	305	16	48	79	5	25	285
43	O	8	39	9	53	9	40	40	6	10	22
44	S	6	41	10	209	14	164	79	8	42	284
45	O	7	37	8	45	9	35	45	5	11	27
46	S	7	36	8	64	75	238	83	8	15	287
47	O	6	37	9	51	9	39	40	3	12	29
48	S	8	43	10	66	13	182	528	5	107	287
49	O	6	32	5	38	6	29	42	4	12	28
50	S	7	38	61	348	15	39	647	5	17	284
51	O	7	39	7	49	10	37	43	4	11	28
52	S	8	42	7	146	18	107	412	38	112	281
53	O	6	41	11	55	8	37	35	4	13	29
54	S	7	37	8	336	15	92	55	5	36	287
55	O	7	36	9	49	8	37	35	3	11	21
56	S	6	38	7	62	14	91	58	4	16	279
57	O	7	38	7	46	8	35	39	5	11	30
58	S	6	39	8	233	41	40	445	66	13	284
59	O	5	32	6	49	8	41	42	7	13	27
60	S	6	37	31	249	13	46	112	65	108	284
61	O	6	38	9	51	10	36	41	4	14	25
62	S	6	42	9	274	82	86	169	5	18	276
63	O	6	39	8	41	11	36	43	4	9	24
64	S	7	42	7	220	51	173	54	4	17	279
65	O	7	35	8	39	8	38	41	4	12	29
66	S	6	34	7	284	91	63	1213	25	18	285
67	O	6	33	8	45	8	35	38	4	11	27
68	S	6	35	10	212	14	41	538	8	21	285
69	O	7	38	7	49	8	30	37	3	11	30
70	S	7	41	11	236	16	86	774	309	288	281
71	O	8	36	8	52	10	34	37	3	10	25

Suite sur la prochaine page...

Tableau A.3 – Suite de la page précédente

Séquence	Mode	P1 (s)	P2 (s)	P3 (s)	P4 (s)	P5 (s)	P6 (s)	P7 (s)	P8 (s)	P9 (s)	P10 (s)
72	S	7	40	7	235	100	238	154	24	19	286
73	O	6	28	7	50	9	37	41	4	11	27
74	S	6	29	6	255	13	71	58	4	28	282
75	O	6	27	6	35	7	32	42	3	11	28
76	S	7	41	9	277	54	78	169	5	19	286
77	O	6	39	8	41	6	30	42	3	11	28
78	S	7	42	13	171	101	156	7427	48	54	272
79	O	6	33	7	42	9	35	40	5	11	30
80	S	7	44	10	237	66	158	116	9	58	282
81	O	6	39	8	46	8	33	34	3	9	22
82	S	8	37	59	57	14	170	75	7	29	282
83	O	5	34	9	52	9	30	36	4	11	32
84	S	6	38	8	236	75	213	79	5	15	276
85	O	5	35	9	48	10	35	38	4	12	29
86	S	6	40	8	248	87	106	1234	263	139	286
87	O	6	38	8	42	7	32	40	6	11	27
88	S	8	42	67	256	12	48	47	6	13	283
89	O	7	38	7	51	6	36	37	4	9	24
90	S	6	40	9	225	77	45	46	4	15	281
91	O	7	28	7	37	6	36	37	3	11	24
92	S	7	43	8	237	14	39	53	8	18	282
93	O	7	39	7	50	9	37	38	5	11	29
94	S	6	39	71	253	14	39	979	9	26	280
95	O	6	39	8	48	8	38	41	3	11	26
96	S	6	37	6	205	14	50	79	4	24	284
97	O	6	31	6	47	9	36	39	3	13	29
98	S	7	38	8	234	35	138	77	7	18	285
99	O	5	29	6	39	7	30	36	3	13	28
100	S	6	39	10	192	15	101	113	6	15	281

Tableau A.4 – Compilation des sabotages

DN	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	Total	Rang	Pourcentage (%)
DN1	0	0	0	0	0	0	0	0	0	0	0	-	0.0
DN2	9	3	8	11	5	7	14	9	13	7	86	3	17.2
DN3	10	9	7	6	11	11	5	6	5	11	81	4	16.2
DN4	8	12	11	13	9	8	7	9	10	5	92	2	18.4
DN5	6	11	8	5	6	9	8	8	4	6	71	6	14.2
DN6	8	8	7	7	13	8	11	8	10	14	94	1	18.8
DN7	9	7	9	8	6	7	5	10	8	7	76	5	15.2
Total	50	50	50	50	50	50	50	50	50	50	-	-	100

Tableau A.5 – Nodes affectées lors des exécutions en mode Sabotages

SEQ	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
2	DN02	DN07	DN04	DN02	DN06	DN03	DN04	DN05	DN06	DN07
4	DN06	DN07	DN07	DN02	DN02	DN03	DN06	DN02	DN05	DN07
6	DN06	DN07	DN06	DN06	DN04	DN03	DN06	DN03	DN02	DN02
8	DN07	DN07	DN03	DN07	DN05	DN04	DN06	DN04	DN06	DN03
10	DN03	DN02	DN05	DN06	DN06	DN04	DN02	DN07	DN03	DN07
12	DN07	DN07	DN07	DN03	DN04	DN06	DN06	DN06	DN02	DN03
14	DN04	DN03	DN05	DN04	DN05	DN07	DN04	DN07	DN07	DN06
16	DN07	DN06	DN04	DN02	DN03	DN04	DN06	DN07	DN07	DN04
18	DN03	DN07	DN06	DN04	DN05	DN02	DN02	DN04	DN06	DN03
20	DN03	DN05	DN07	DN06	DN04	DN02	DN02	DN03	DN07	DN02
22	DN05	DN03	DN05	DN05	DN03	DN07	DN04	DN05	DN02	DN05
24	DN06	DN06	DN02	DN02	DN06	DN03	DN03	DN06	DN02	DN06
26	DN02	DN04	DN07	DN03	DN06	DN03	DN05	DN04	DN02	DN02
28	DN07	DN03	DN03	DN03	DN07	DN05	DN02	DN02	DN07	DN05
30	DN02	DN04	DN05	DN06	DN07	DN03	DN04	DN04	DN06	DN04
32	DN03	DN05	DN02	DN07	DN06	DN05	DN05	DN06	DN06	DN06
34	DN06	DN02	DN05	DN02	DN06	DN06	DN02	DN02	DN02	DN06

Suite sur la prochaine page...

Tableau A.5 – suite de la page précédente

SEQ	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
36	DN05	DN06	DN04	DN02	DN03	DN06	DN02	DN02	DN04	DN07
38	DN02	DN07	DN03	DN04	DN03	DN05	DN07	DN02	DN02	DN02
40	DN06	DN03	DN05	DN06	DN03	DN03	DN05	DN02	DN02	DN05
42	DN07	DN03	DN04	DN03	DN04	DN07	DN03	DN05	DN05	DN04
44	DN03	DN05	DN04	DN04	DN06	DN05	DN07	DN05	DN06	DN06
46	DN04	DN03	DN05	DN04	DN03	DN07	DN06	DN03	DN04	DN06
48	DN07	DN04	DN04	DN04	DN02	DN06	DN05	DN05	DN07	DN03
50	DN06	DN05	DN02	DN07	DN03	DN03	DN03	DN07	DN02	DN04
52	DN05	DN04	DN06	DN04	DN06	DN02	DN03	DN04	DN05	DN05
54	DN07	DN04	DN03	DN06	DN07	DN03	DN04	DN07	DN05	DN05
56	DN02	DN05	DN02	DN07	DN07	DN03	DN07	DN06	DN06	DN06
58	DN07	DN04	DN04	DN04	DN07	DN02	DN02	DN04	DN07	DN03
60	DN02	DN05	DN07	DN02	DN06	DN06	DN04	DN03	DN07	DN07
62	DN03	DN05	DN06	DN04	DN07	DN05	DN06	DN06	DN03	DN04
64	DN04	DN04	DN04	DN05	DN02	DN04	DN02	DN05	DN06	DN06
66	DN07	DN06	DN04	DN03	DN06	DN04	DN06	DN07	DN07	DN02
68	DN05	DN04	DN07	DN07	DN05	DN07	DN02	DN05	DN04	DN06
70	DN02	DN05	DN06	DN04	DN04	DN04	DN02	DN07	DN04	DN05
72	DN05	DN06	DN04	DN07	DN03	DN02	DN02	DN04	DN02	DN03
74	DN03	DN02	DN06	DN02	DN02	DN04	DN06	DN02	DN04	DN06
76	DN06	DN03	DN03	DN05	DN02	DN07	DN04	DN03	DN04	DN06
78	DN04	DN06	DN02	DN07	DN04	DN06	DN05	DN07	DN04	DN06
80	DN03	DN04	DN07	DN04	DN06	DN04	DN02	DN07	DN03	DN03
82	DN03	DN05	DN02	DN02	DN04	DN06	DN07	DN05	DN03	DN06
84	DN04	DN06	DN06	DN05	DN06	DN02	DN05	DN02	DN02	DN07
86	DN05	DN03	DN07	DN04	DN03	DN05	DN02	DN03	DN06	DN03
88	DN06	DN03	DN02	DN05	DN06	DN07	DN07	DN06	DN06	DN03
90	DN03	DN04	DN04	DN06	DN05	DN05	DN06	DN06	DN04	DN03
92	DN04	DN06	DN03	DN03	DN03	DN05	DN05	DN02	DN02	DN06
94	DN04	DN05	DN07	DN02	DN04	DN03	DN06	DN07	DN04	DN02
96	DN02	DN04	DN02	DN02	DN03	DN05	DN03	DN06	DN02	DN07
98	DN02	DN05	DN05	DN07	DN05	DN06	DN05	DN04	DN03	DN03
100	DN04	DN04	DN03	DN04	DN04	DN02	DN02	DN04	DN04	DN02

TABLEAU A.6 – Valeurs des métriques (variables) utilisées en mode Sabotages

SEQ	MODE	S1 (%)	S1 (s)	S2 (Mo)	S2 (s)	S3 (Go)	S5 (ms)	S5 (ms)	S6 (%)	S6 (%)	S8 (s)	S9 (%)
2	S	54	57	1052	115	6	60	16	26	17	71	2
4	S	65	43	1043	53	6	54	14	28	11	94	4
6	S	52	56	1572	55	6	84	12	23	14	55	8
8	S	58	56	1392	54	7	60	11	26	14	116	4
10	S	77	67	1772	39	6	69	14	19	11	64	4
12	S	70	77	1855	41	7	90	14	19	15	57	7
14	S	78	67	1767	114	6	99	12	18	18	48	8
16	S	51	52	1579	44	7	66	16	27	16	114	4
18	S	67	43	1057	66	6	57	14	27	14	74	8
20	S	56	37	1593	90	6	91	15	26	10	113	7
22	S	50	75	1975	80	6	98	18	27	12	64	10
24	S	74	66	1142	72	7	80	16	26	15	95	5
26	S	77	113	2027	55	7	99	12	19	12	91	6
28	S	61	47	1338	103	6	56	10	27	13	111	6
30	S	55	43	1827	119	7	77	11	23	10	40	1
32	S	78	58	1753	36	6	97	18	15	17	62	3
34	S	70	87	1899	30	7	59	12	19	14	106	7
36	S	73	109	1181	95	6	83	14	23	16	115	6
38	S	58	112	1926	41	7	63	18	17	15	81	7
40	S	77	88	1431	56	6	53	10	16	18	49	3
42	S	77	76	1895	80	7	83	14	29	14	31	10
44	S	51	50	1103	117	6	89	10	23	19	68	8
46	S	53	92	1698	48	7	77	17	16	17	65	3
48	S	64	98	1231	113	6	59	19	29	10	106	10
50	S	53	40	1026	94	7	90	15	28	11	86	9
52	S	65	64	1677	60	7	95	14	19	10	39	9
54	S	75	83	1836	38	7	71	15	20	18	109	10
56	S	55	58	1774	63	7	90	15	23	11	65	6
58	S	56	98	1075	103	6	76	18	19	14	82	2
60	S	65	66	1731	50	7	92	14	16	10	89	4

Suite sur la prochaine page...

Tableau A.6 – suite de la page précédente

SEQ	MODE	S1 (%)	S1 (s)	S2 (Mo)	S2 (s)	S3 (Go)	S5 (ms)	S5 (ms)	S6 (%)	S6 (%)	S8 (s)	S9 (%)
62	S	65	104	1725	30	7	77	12	15	19	61	6
64	S	53	41	1383	101	6	98	16	27	15	32	10
66	S	68	65	1835	75	6	92	11	16	15	105	3
68	S	50	93	1064	38	6	55	14	24	12	103	1
70	S	55	45	1310	87	6	82	13	25	14	92	4
72	S	71	109	1682	66	7	75	19	27	19	54	3
74	S	69	47	1620	55	6	72	18	18	15	70	10
76	S	61	91	1107	78	7	83	17	23	13	79	2
78	S	55	116	1587	73	7	55	19	16	10	70	6
80	S	68	45	1211	86	6	88	10	28	13	92	1
82	S	55	67	1466	45	7	64	13	19	19	94	10
84	S	75	35	1096	119	7	71	18	22	10	75	1
86	S	55	45	1353	73	6	81	10	18	10	102	9
88	S	52	32	1618	55	7	56	15	16	19	90	6
90	S	59	105	1586	92	7	99	18	22	12	92	1
92	S	64	84	1088	75	6	73	11	25	13	92	6
94	S	52	89	1760	72	6	56	11	15	16	32	1
96	S	57	82	1207	67	7	85	12	29	13	78	3
98	S	60	100	1169	114	6	72	14	19	13	107	6
100	S	69	49	1524	97	6	98	18	25	13	71	8

Annexe B

OZONE CLUSTER

L'extrait de code B.1 présente des exemples de création, finalisation et fermeture de *pipeline*s provenant du journal du SCM.

L'extrait de code B.2 présente le résultat de la commande Ozone permettant de lister les *nodes* et leurs *pipeline*s : *ozone admin datanode list*

L'extrait de code B.3 présente le résultat du processus d'élection du *leader* de RAFT dans Ozone.

EXTRAIT DE CODE B.1 – Ozone - *Pipelines*

```
1 # LOG DU SCM
2 [RatisPipelineUtilsThread - 0] INFO
  ↪ org.apache.hadoop.hdds.scm.pipeline.RatisPipelineProvider:
  ↪ Sending CreatePipelineCommand for
  ↪ pipeline:PipelineID=746abc63-1e7e-485d-a65a-fccd7e036da5 to
  ↪ datanode:b24bcb3c-7b0d-4564-a324-07315c38939d
3
4 [EventQueue-StaleNodeForStaleNodeHandler] INFO
  ↪ org.apache.hadoop.hdds.scm.node.StaleNodeHandler: Datanode
  ↪ 0d1365d8-5355-494e-a061-b14a51337e32{ip: 000.000.000.133,
  ↪ host: servozone04, ports: [REPLICATION=9886, RATIS=9858,
  ↪ RATIS_ADMIN=9857, RATIS_SERVER=9856, STANDALONE=9859],
  ↪ networLocation: /default-rack, certSerialId: null,
  ↪ persistedOpState: IN_SERVICE, persistedOpStateExpiryEpochSec:
  ↪ 0} moved to stale state. Finalizing its pipelines
  ↪ [PipelineID=cd9153ab-5a9b-4112-ae85-0b167c69c418,
  ↪ PipelineID=94204635-00ac-4734-82c5-b6c0b50250e5]
5
```

```
6 [EventQueue-StaleNodeForStaleNodeHandler] INFO
  ↪ org.apache.hadoop.hdds.scm.pipeline.PipelineManagerImpl:
  ↪ Pipeline Pipeline[ Id: 94204635-00ac-4734-82c5-b6c0b50250e5,
  ↪ Nodes: 0d1365d8-5355-494e-a061-b14a51337e32{ip:
  ↪ 000.000.000.133, host: servozone04, ports: [REPLICATION=9886,
  ↪ RATIS=9858, RATIS_ADMIN=9857, RATIS_SERVER=9856,
  ↪ STANDALONE=9859], networkLocation: /default-rack,
  ↪ certSerialId: null, persistedOpState: IN_SERVICE,
  ↪ persistedOpStateExpiryEpochSec:
  ↪ 0}5f2009cd-53bf-4d5e-ba6b-b0005082720e{ip: 000.000.000.130,
  ↪ host: servozone01, ports: [REPLICATION=9886, RATIS=9858,
  ↪ RATIS_ADMIN=9857, RATIS_SERVER=9856, STANDALONE=9859],
  ↪ networkLocation: /default-rack, certSerialId: null,
  ↪ persistedOpState: IN_SERVICE, persistedOpStateExpiryEpochSec:
  ↪ 0}a9522048-a43c-4cdd-97fd-8bb12bb2d816{ip: 000.000.000.135,
  ↪ host: servozone06, ports: [REPLICATION=9886, RATIS=9858,
  ↪ RATIS_ADMIN=9857, RATIS_SERVER=9856, STANDALONE=9859],
  ↪ networkLocation: /default-rack, certSerialId: null,
  ↪ persistedOpState: IN_SERVICE, persistedOpStateExpiryEpochSec:
  ↪ 0}, ReplicationConfig: RATIS/THREE, State:OPEN,
  ↪ leaderId:a9522048-a43c-4cdd-97fd-8bb12bb2d816,
  ↪ CreationTimestamp2023-08-11T10:43:38.585794-04:00[America/New_York]]
  ↪ moved to CLOSED state
```

EXTRAIT DE CODE B.2 – Ozone - Liste des *nodes* et leurs *pipeline*s avant une exécution

```
~/ozone-1.3.0/bin$ ./ozone admin datanode list
Datanode: 80e089e6-e253-4247-83f6-718112f0c474
 ↪ (/default-rack/000.000.000.132/servozone03/2 pipelines)
Operational State: IN_SERVICE
Health State: HEALTHY
Related pipelines:
c528aa0f-78b6-4bed-91df-a8c0cc508fd7/RATIS/THREE/RATIS/OPEN/Follower
b24ca90e-0ba8-4070-adaf-a6e06fce874e/RATIS/THREE/RATIS/OPEN/Leader

Datanode: a9522048-a43c-4cdd-97fd-8bb12bb2d816
 ↪ (/default-rack/000.000.000.135/servozone06/1 pipelines)
Operational State: IN_SERVICE
Health State: HEALTHY
Related pipelines:
bf98b064-ee36-4871-b76b-26358d0ff74f/RATIS/THREE/RATIS/OPEN/Leader

Datanode: b24bcb3c-7b0d-4564-a324-07315c38939d
 ↪ (/default-rack/000.000.000.136/servozone07/2 pipelines)
Operational State: IN_SERVICE
Health State: HEALTHY
Related pipelines:
c528aa0f-78b6-4bed-91df-a8c0cc508fd7/RATIS/THREE/RATIS/OPEN/Follower
b24ca90e-0ba8-4070-adaf-a6e06fce874e/RATIS/THREE/RATIS/OPEN/Follower

Datanode: 0d1365d8-5355-494e-a061-b14a51337e32
 ↪ (/default-rack/000.000.000.133/servozone04/2 pipelines)
Operational State: IN_SERVICE
Health State: HEALTHY
Related pipelines:
c528aa0f-78b6-4bed-91df-a8c0cc508fd7/RATIS/THREE/RATIS/OPEN/Leader
69c736aa-4702-4a00-94bd-e1991482bc61/RATIS/THREE/RATIS/OPEN/Follower

```

```
29 Datanode: 41b29633-d924-4b37-bc10-fc9557bd95d8
   ↪ (/default-rack/000.000.000.131/servozone02/2 pipelines)
30 Operational State: IN_SERVICE
31 Health State: HEALTHY
32 Related pipelines:
33 bf98b064-ee36-4871-b76b-26358d0ff74f/RATIS/THREE/RATIS/OPEN/Follower
34 69c736aa-4702-4a00-94bd-e1991482bc61/RATIS/THREE/RATIS/OPEN/Leader
35
36 Datanode: 4a12af57-593b-4f0e-9a2e-d8dc9a5a5231
   ↪ (/default-rack/000.000.000.134/servozone05/2 pipelines)
37 Operational State: IN_SERVICE
38 Health State: HEALTHY
39 Related pipelines:
40 b24ca90e-0ba8-4070-adaf-a6e06fce874e/RATIS/THREE/RATIS/OPEN/Follower
41 69c736aa-4702-4a00-94bd-e1991482bc61/RATIS/THREE/RATIS/OPEN/Follower
42
43 Datanode: 5f2009cd-53bf-4d5e-ba6b-b0005082720e
   ↪ (/default-rack/000.000.000.130/servozone01/1 pipelines)
44 Operational State: IN_SERVICE
45 Health State: HEALTHY
46 Related pipelines:
47 bf98b064-ee36-4871-b76b-26358d0ff74f/RATIS/THREE/RATIS/OPEN/Follower
```

EXTRAIT DE CODE B.3 – Ozone - RAFT - Élection

```
# LOG DE SERVOZONE01 - SÉQUENCE #100
* 5f2009cd-53bf-4d5e-ba6b-b0005082720e - SERVOZONE01
* 4a12af57-593b-4f0e-9a2e-d8dc9a5a5231 - SERVOZONE05
* 41b29633-d924-4b37-bc10-fc9557bd95d8 - SERVOZONE02

[5f2009cd-53bf-4d5e-ba6b-b0005082720e@group-EBEAC28D20DE-LeaderElection68]
    ratis.server.impl.LeaderElection:
    5f2009cd-53bf-4d5e-ba6b-b0005082720e@group-EBEAC28D20DE-LeaderElection68
    ELECTION round 0: submit vote requests at term 11 for -1:
    peers:

[5f2009cd-53bf-4d5e-ba6b-b0005082720e|rpc:000.000.000.130:9856|
startupRole:FOLLOWER,
    4a12af57-593b-4f0e-9a2e-d8dc9a5a5231|rpc:000.000.000.134:9856|
startupRole:FOLLOWER,
    41b29633-d924-4b37-bc10-fc9557bd95d8|rpc:000.000.000.131:9856|
startupRole:FOLLOWER]


[5f2009cd-53bf-4d5e-ba6b-b0005082720e@group-EBEAC28D20DE-LeaderElection68]
ratis.server.RaftServerConfigKeys:
    raft.server.rpc.first-election.timeout.min = 5s (fallback to
    raft.server.rpc.timeout.min)
ratis.server.RaftServerConfigKeys:
    raft.server.rpc.first-election.timeout.max = 5200ms (fallback
    to raft.server.rpc.timeout.max)
ratis.server.impl.LeaderElection:
    5f2009cd-53bf-4d5e-ba6b-b0005082720e@group-EBEAC28D20DE-LeaderElection68:
    ELECTION PASSED received 1 response(s) and 0 exception(s):
ratis.server.impl.LeaderElection: Response 0:
    5f2009cd-53bf-4d5e-ba6b-b0005082720e <-
    41b29633-d924-4b37-bc10-fc9557bd95d8# 0:OK-t11
```

```
19 ratis.server.impl.LeaderElection:
   ↪ 5f2009cd-53bf-4d5e-ba6b-b0005082720e@group-EBEAC28D20DE -
   ↪ LeaderElection68 ELECTION round 0: result PASSED
20 ratis.server.impl.RoleInfo: 5f2009cd-53bf-4d5e-ba6b-b0005082720e:
   ↪ shutdown
   ↪ 5f2009cd-53bf-4d5e-ba6b-b0005082720e@group-EBEAC28D20DE-LeaderElection68
21 ratis.server.RaftServer$Division:
   ↪ 5f2009cd-53bf-4d5e-ba6b-b0005082720e@group-EBEAC28D20DE:
   ↪ changes role from CANDIDATE to LEADER at term 11 for
   ↪ changeToLeader
22 hadoop.ozone.container.common.transport.server.ratis.XceiverServerRatis:
   ↪ Leader change notification received for group:
   ↪ group-EBEAC28D20DE with new leaderId:
   ↪ 5f2009cd-53bf-4d5e-ba6b-b0005082720e
23 ratis.server.RaftServer$Division:
   ↪ 5f2009cd-53bf-4d5e-ba6b-b0005082720e@group-EBEAC28D20DE:
   ↪ change Leader from null to
   ↪ 5f2009cd-53bf-4d5e-ba6b-b0005082720e at term 11 for
   ↪ becomeLeader, leader elected after 55715ms
```