République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université A/Mira de Béjaia
Faculté des Sciences Exactes
Département d'Informatique

MÉMOIRE DE MASTER RECHERCHE

En
Informatique

Option
Réseaux et Systèmes Distribués

Thème

Gestion de la Cohérence des données Répliquées dans le Cloud

Présenté par : M^{lle} Ait Ali Sylia et M^{lle} Lahdir Meriem

Soutenu le 30 Juin 2016 devant le jury composé de :

Président	**Dr K. Adel**	**Maître de conf. A**	**U. A/Mira Béjaia.**
Examinateur	**M. R. Ouzeggane**	**Maître de conf. B**	**U. A/Mira Béjaia.**
Examinatrice	**M^{lle} L. Alouache**	**Doctorante "LMD"**	**U. Sétif.**
Encadrant	**M^{e} Yaici**	**Maître de conf. B**	**U. A/Mira Béjaia.**

Béjaia, Juin 2016.

⁕ *Remerciements* ⁕

Nos remerciements s'adressent à DIEU pour nous avoir donné le courage et la détermination nécessaires pour finaliser ce travail.

Nos remerciements s'adressent également à Mme YAICI, qui a bien accepté de diriger notre travail. Ses remarques, ses précieux conseils et ses corrections nous ont été d'une grande utilité. Nous lui disons " Merci ".

Nous tenons à remercier les membres de notre jury, pour avoir accepté de juger notre travail. Leurs remarques et suggestions nous seront très utiles pour la finalisation de ce mémoire.

Nous exprimons notre profonde gratitude à nos parents, qui ont fait de nous ce que nous sommes aujourd'hui. Ils nous ont beaucoup apporté tout au long de notre existence, ils nous ont offerts les moyens et tous les efforts qui vont avec pour nous voir réussir dans nos vies.

Nous remercions profondément nos frères, qui nous ont soutenus et conseillés durant toutes ces années d'études.

Nos plus vifs remerciements s'adressent aussi à tous nos amis et grandes familles, qui nous ont soutenus et épaulés tout au long de la réalisation de ce travail, puissent-ils être assurés que chacun d'eux est présent dans nos esprits.

Nous exprimons notre profonde gratitude à toute personne qui, de près ou de loin, a contribué à l'élaboration de ce travail.

⁂ *Dédicaces* ⁂

A nos parents, nos êtres chers ;
A nos frères, nos conseillés, nos protecteurs ;
A toutes nos familles, nos bien-aimés ;
A tous nos amis, Asma, Sofie, Yasmine, et à tous ceux que nous ne pouvons citer à défaut d'espace, vous qui êtes nos alliés , nos complices, nos confidents ;
A vous tous, qui lisez ce mémoire ;

De : Sylia , Meriem

Table des matières

Table des figures

Liste des tableaux

Notations et symboles

ACID	**A**vailability **C**onsistency **I**solation **D**urability
API	**A**pplication **P**rogramation **I**nterface
AWS	**A**mazon **W**eb **S**ervice
BDD	**B**ases **D**e **D**onnées
C3	**C**ost-based **C**oncurrency **C**ontrol
CCP	**C**oncurrency **C**ontrol **P**rotocol
CH	**C**luster**H**ead
CPU	**C**ontrol **P**rocessing **U**nit
CRDT	**C**ommutative **R**eplicated **D**ata **T**ype
DaaS	**D**ata **a**s **a** **S**ervice
DCaaS	**D**ata **C**onsistency **a**s **a** **S**ervice
EC2	**E**lastic **C**loud **C**ompute
HRD	**H**igh **R**eplication **D**atastore
HPC	**H**igh **P**erformance **C**omputing
IaaS	**I**nfastructure **a**s **a** **S**ervice
ID	**ID**entifiant
IHM	**I**nterface **H**omme-**M**achine
IP	**I**nternet **P**rotocol
JADE	**J**ava **A**gent **DE**velopement Framework
LibRe	**Lib**rary for **Re**plication
LMT	**L**ast **M**odified **T**ime
LS	**L**og **S**tate
MATLAB	**MAT**rix **LAB**oratory
MCL	**M**arkov **Cl**uster algorithm
MESI	**M**odified **E**xclusive **S**hared **I**nvalid
NoSQL	**N**ot **o**nly **S**tructured **Q**uery **L**anguage
PaaS	**P**latform **a**s **a** **S**ervice
PFS	**P**arallel **F**ile **S**ystem
PPS	**P**artial **P**ersistent **S**tructure
RAID	**R**edundant **A**rray of **I**ndependent **D**isks
RE	**R**equête d'**E**criture
RL	**R**equête de **L**ecture
SaaS	**S**oftware **a**s **a** **S**ervice
SGBD	**S**ystème de **G**estion des **B**ases de **D**onnées

SSD	**S**olid-**S**tate **D**rives
VM	**V**irtual **M**achine

Introduction générale

La taille des données ne cesse d'augmenter et cela de façon exponentielle au sein de nombreuses organisations. En considérant la somme totale des données détenues par tous les grands services de stockage et les entreprises en ligne comme Google, Amazon, Microsoft et Facebook ; Les estimations de la taille des données sont d'au moins 1.200 pétaoctets ; C'est 1,2 million de téraoctets (un téraoctet est de 1000 gigaoctets). Et ce chiffre exclut les autres grands fournisseurs tels que Dropbox, les serveurs massifs dans l'industrie et le milieu universitaire. Les données sont partout et proviennent de sources multiples : les médias sociaux, les téléphones intelligents, les capteurs, etc [9].

Ce tsunami de données, connu sous le nom de Big Data, introduit de nombreuses complications aux différents aspects du stockage et de la gestion des données. Ces complications sont dues à la taille écrasante, mais aussi à la vitesse de traitement requise et la complexité de ces données. Afin de relever les défis connexes, de nombreux systèmes Big Data reposent sur de grandes et de nouvelles infrastructures, ainsi que de nouvelles plates-formes et modèles de programmation. Dans ce contexte, le nouveau paradigme du Cloud Computing offre d'excellents moyens pour Big Data. Le Cloud Computing assure une disponibilité des grands centres de données avec des milliers de périphériques, qui sont reliés par un réseau formant un grand système distribué. Les utilisateurs peuvent ainsi (dans le Cloud) louer des ressources de calcul et de stockage à la demande avec la politique "payez ce que vous consommez". Ainsi, les entreprises peuvent acquérir les ressources nécessaires à leurs applications Big Data à un faible coût en cas de besoin. Pendant ce temps, ils évitent de gros investissements sur les infrastructures physiques qui ont besoin d'énormes efforts pour leur construction et leur entretien.

Pour garantir une haute disponibilité face aux défaillances de serveurs et de réseau, et répondre à la demande croissante aux requêtes des clients, des systèmes de gestion de données Cloud répliquent des données sur plusieurs centres de données permettant ainsi un accès global continu. Cependant, assurer la cohérence entre ces données répliquées relève

d'un autre niveau. La cohérence forte nécessite d'énormes efforts de synchronisation à travers différents endroits et donc, expose les utilisateurs à des latences de réseau élevées. Cela affecte les performances et la disponibilité dans le Cloud Computing. Une alternative particulière qui est devenu très populaire est la cohérence éventuelle. La cohérence éventuelle peut tolérer la contradiction à certains points dans le temps, mais garantit la convergence de toutes les répliques au même état à une date ultérieure.

De nombreux travaux ont été proposé pour assurer la cohérence de ces répliques dans le Cloud ; parmi ceux que nous avons étudié, ils y en a qui se sont basé sur : les estampilles des données, sur le principe des quorums ou encore ceux qui se sont concentrés sur le coût et les performances des applications.

Notre travail consiste à proposer une solution dans cette optique-là. Assurant ainsi une cohérence éventuelle forte. Et se basant sur les quorums et les estampilles.

Ce mémoire est organisé en quatre chapitres : Dans le premier chapitre, nous introduisons le phénomène Big Data, ainsi que les plates-formes et infrastructures pour faire face aux défis liés. Ensuite, nous faisons un zoom sur le Cloud Computing comme une infrastructure efficace pour la gestion de Big Data, présentant ainsi tous les services qu'offre le Cloud. Pour finir nous avons présenté les avantages de la clusterisation dans le Cloud.

Dans le deuxième chapitre, nous nous sommes familiarisés avec le concept de la réplication et de la cohérence. Nous avons étudié ces deux concepts là dans les Bases de données puis dans la mémoire cache. Ensuite nous avons présenté différents modèles de réplication et de cohérence existants dans la littérature.

Dans le troisième chapitre qui consiste en un état de l'art sur la Cohérence des données répliquées, plusieurs travaux de recherche se rapportant à notre thème ont été étudiés et analysés.

Dans le quatrième chapitre, une solution pour la cohérence des données répliquées dans le Cloud a été proposée. Puis différents scénarios que présente cette solution ont été détaillés grâce à la plateforme JADE. Et enfin, les résultats de l'évaluation des performances de notre solution sont représentés sous Matlab.

Et enfin nous terminerons par une conclusion et des perspectives futurs.

Chapitre 1

Big Data et Cloud

Introduction

L'explosion quantitative (et souvent redondante) de la donnée numérique contraint à de nouvelles manières de voir et analyser le monde. Ce déluge de données, connu sous le nom de "*Big Data*", est tellement volumineux qu'il en devient difficile à manipuler avec des outils classiques de gestion de base de données ou de gestion de l'information. Toutefois, l'effort scientifique a abouti à un environnement capable à la fois de le contenir et de le gérer, appelé *Cloud*.

1.1 Big Data

1.1.1 Évolution des Big Data

Les données massives ou Big Data, est un terme qui est apparus en 1998. Ce concept, tel qu'il est défini actuellement, englobe un ensemble de technologies et de pratiques destinées à stocker de très grandes masses de données et à les analyser très rapidement. Ce concept est identifié par l'égalité dite *trois V* : Volume, Variété des types et des sources de données, Vélocité (ou vitesse des échanges). Un quatrième V pour Variabilité est apparu ultérieurement [14].

Ces " V" sont définis comme suit [16] :

— **Volume** : Aujourd'hui , la taille des données augmente exponentiellement. Plusieurs entreprises et organisations connaissent le phénoméne *Big Data* en raison de multiples facteurs, tels que les données stockées recueillies au fil des ans, la diffusion et le partage de données sur les réseaux sociaux, l'augmentation des données colléctées grâce aux capteurs, etc.

— **Variété** : La prolifération de types de données provenant de diverses sources comme

les médias sociaux, les interactions entre les sites et les terminaux mobiles, crée une très grande diversité au-delà des données transactionnelles traditionnelles. Les données ne s'inscrivent plus dans des structures nettes, faciles à consommer.

— **Vélocité (Vitesse)** : Pour de nombreuses organisations, l'aspect le plus difficile du Big Data n'est pas exclusivement le volume mais plutôt la vitesse de traitement des données pour répondre aux demandes des utilisateurs. Un large éventail d'applications nécessite un traitement de données rapide en quasi-temps réel, peu importe la taille des données. Par exemple, le commerce électronique, le placement d'annonces sur les pages Web, etc.

— **Variabilité** : En plus de la vitesse et de la variété , les données peuvent présenter une grande variabilité dans leurs formats et leur contenu et cela peut se traduire par des données incohérentes ou des flux de données non réguliers avec des pics périodiques.

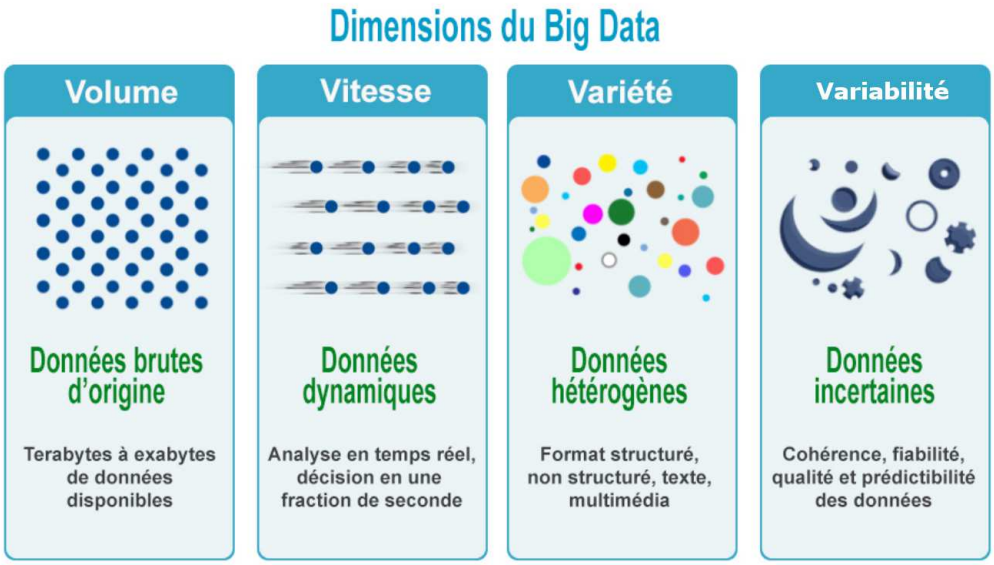

FIGURE 1.1 – Dimensions du Big Data [14].

1.1.2 Types de Big Data

— **Métadonnées générées par la plateforme** : Ce sont des informations sur les données introduites, qui sont générées instantanément par la machine, comme la date et le numéro de la mise á jour.

— **Métadonnées générées par le client** : Le client, lors de l'introduction de ses données, introduit aussi des informations les concernants, comme la sutructuration (nom de fichier, chapitre, section, partie...etc).

— **Données générées par le client** : Ce sont les données que l'Homme introduit (livres, musique...).

1.1.3 Technologies Big Data

La croissance des volumes de données collectées, essentiellement de type non structuré, a posé les problématiques de leur stockage, de leur traitement et de leur analyse à des fins de prise de décision.

Pour faire face à ces problématiques, des technologies ont été implémentées, nous citons :

1.1.3.1 Bases de données NoSQL

L'absence de relation évidente ou de classement possible est un point important qui annonce la difficulté de recourir à des structures bien définies comme celles des bases de données relationnelles. C'est pourquoi la catégorie NoSQL est créée, elle désigne l'ensemble des données pouvant être intérogées grâce aux requêtes ou données structurées, et celles qui ne sont pas structurées.

D'après cette définition, les bases de données NoSQL diffèrent des bases SQL dans la mesure où il y a absence de schéma statique prédéfini (relationnel) structurant les données ; la qualification de semi-structurées qui peut être perçue comme un manque peut ainsi être perçue comme une fonctionnalité plus riche ou de moins plus souple [14].

1.1.3.2 Systèmes de fichiers parallèles

Un système de fichiers parallèles (connu sous PFS pour Parallel File System) a pour but de permettre l'accès simultané à un système de fichiers à plusieurs clients. Ce qui le distingue d'un simple système de fichiers partagé est le parallélisme au niveau :

— **Des clients** : Plusieurs clients peuvent lire et écrire simultanément et non pas chacun à son tour.
— **De la répartition des données** : Un client l'utilisant profitera de bonnes performances si les données sont réparties sur plusieurs serveurs de données.

Ce parallélisme se fait de façon transparente pour le client qui voit le système de fichiers comme s'il était local. En plus des fonctions d'un système de fichier local, un système de fichiers parallèle doit gérer efficacement les éventuels conflits entre les différents clients. L'approche privilégiée consiste à utiliser des verrous pour limiter/contrôler les accès simultanés à un fichier ou répertoire donné [21].

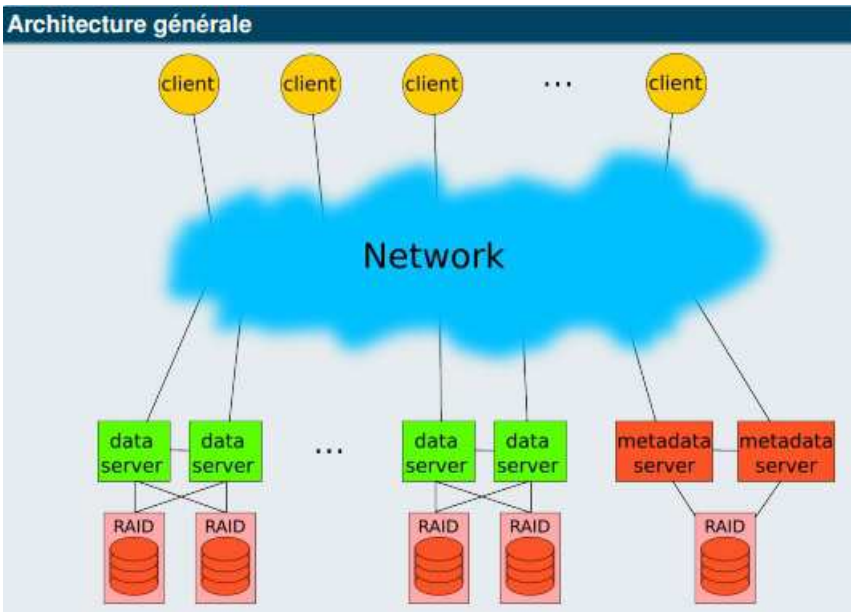

FIGURE 1.2 – Principe des PFS [21].

1.1.3.3 MapReduce

MapReduce est un modèle de programmation présenté par Google afin de fournir d'une façon massivement parallèle le traitement efficace des Big Data. Ce modèle fonctionne selon deux étapes : Map et Reduce [21] :

1. **Map** : un job est soumis par un nœud au nœud contenant les métadonnées ; celui-ci délègue les tâches à effectuer à d'autres nœuds sur la base d'une paire (clé, valeur).
2. **Reduce** : les nœuds les plus bas font remonter leurs résultats au nœud parent qui les avaient sollicités ; celui-ci associe avec la fonction Reduce toutes les valeurs correspondant à la même clé en une paire unique (clé, valeur) et le processus se poursuit si plusieurs niveaux hiérarchiques ont été actives. A la fin du job le nœud demandeur recompose la réponse globale à la requête.

Un traitement Map Reduce ne se limite pas en général à un seul travail, c'est un enchaînement de travaux qui necessite un ordonnanceur tel qu'Oozie [14].

1.1.3.4 Hadoop

Hadoop est un projet de Apache Software Foundation dont l'objectif est de développer des logiciels open source pour des traitements de données fiables, extensibles et distribués. Elle met à disposition des bibliothèques, des programmes permettant le traitement de volume

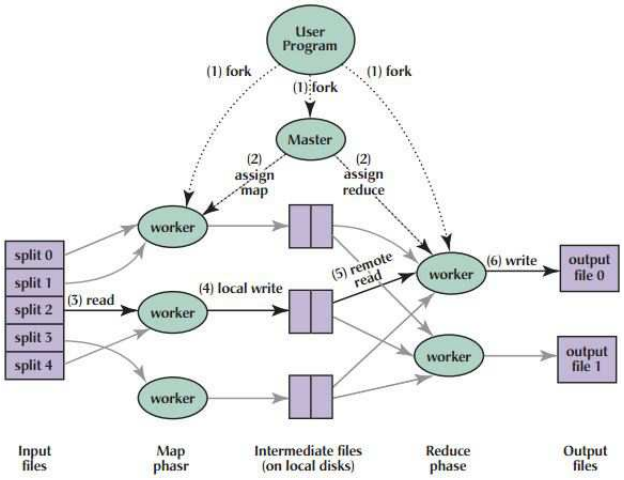

FIGURE 1.3 – Map Reduce [36].

important de données au moyen des clusters de serveurs en utilisant un modèle simple de programmation MapReduce pour l'extraction des données stockées. Hadoop est un environnement de logiciels utilisés par différents acteurs du monde de l'informatique pour développer leurs propres solutions [21].

1.2 Cloud Computing

Après avoir détaillé les grands points concernant les Big Data, nous allons expliciter la platforme la mieux adapté pour le traitement de ces données : le Cloud.

1.2.1 Un nouveau modèle de référence

Le Cloud est un ensemble de ressources informatiques reliées par le protocole réseau internet (IP), dont la taille est totalement élastique. L'ensemble de ces ressources est utilisable pour présenter des services complexes de façon simple aux clients. La multiplicité des ressources permet de proposer des services à valeur ajoutée pour un coût assez bas.

On retrouve dans cette description les architectures antérieures (client-serveur, orienté service) avec une augmentation de données provenant de l'évolution des couches logicielles reliant les applications aux ressources disponibles. On y retrouve une qualité essentielle de ce modèle, l'extensibilité, qui était également caractéristiques des PFS. La simplicité d'utilisation et d'accès aux Clouds se traduit par le fait qu'ils s'adressent aussi bien aux entreprises et aux administrations qu'au grand public. Les objectifs de coûts peu élevés font dire à certains que les Clouds réalisent l'objectif de démocratisation du calcul et du stockage informatique.

On trouve également dans cette description une caractéristique absente des PFS qui est la distribution des données sur de grandes distances [14].

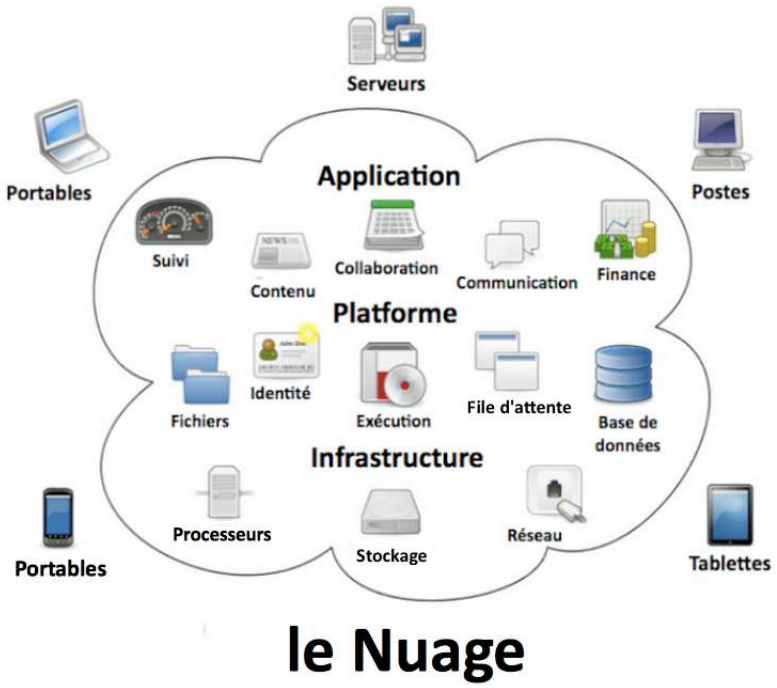

FIGURE 1.4 – Services du Cloud [16].

1.2.2 Niveaux de services qu'offre le Cloud

Il existe trois niveaux de services offert par le Cloud, selon le besoin du client : Infrastructure as a Service (IaaS), Platform as a Service (PaaS), Software as a Service (SaaS) [16].

— **Infrastructure as a Service (IaaS)** : Dans ce niveau de service, les clients obtiennent l'infrastructure et les équipements nécessaires pour l'implémentation de leurs plateformes ou applications. Ceci, en louant sur demande les machines virtuelles (VMs), et les ressources de stockage dont ils ont besoin. Le fournisseur du Cloud héberge, gère, et maintient l'ensemble des équipements d'infrastructure incluant le stockage, les serveurs, le routage et le matériel. Ainsi, la tâche de la gestion et la maintenance du logiciel revient au client.

— **Platform as a Service (PaaS)** : En plus des équipements fournis par le niveau cité précédemment, ce niveau de service y ajoute la plateforme (système d'exploitation), dont les utilisateurs (clients) se servent pour l'exécution de leurs applications.

— **Software as a Service (SaaS)** : C'est le plus haut niveau de la hiérarchie. Dans ce modèle, les logiciels sont hébergés dans l'infrastructure du Cloud fournisseur, puis délivrés à travers un réseau, Internet spécialement. Ce type de service est bénéfique pour le client puisqu'il facilite l'administration des applications, permet une meilleure homogénéité des versions entre les applications utilisateurs, et assure un accès global et les collaborations.

1.2.3 Modèles de déploiement du Cloud

Il existe quatre modèles de déploiement du Cloud, selon le besoin des entreprises clientes :

— **Cloud Public** : Les utilisateurs n'ont aucun contrôle de l'infrastructure ni aucun pouvoir d'audit. Les fournisseurs mettent à disposition des ressources - par exemple applicatifs ou capacités de stockage - accessibles au moyen de connexions de types Internet ou réseau privé virtuel.Les infrastructures et les logiciels associés sont la propriété des fournisseurs. L'utilisateur paie pour les services consommés grâce à des moyens appropriés de facturation. La sécurité est moins élevée que dans le Cloud privé.

— **Cloud Privé** : Cette solution permet le contrôle des ressources et une plus grande capacité à sécuriser l'infrastructure, les applicatifs et les données. L'organisation utilisatrice construit son propre pool de services, hautement virtualisés, et les rend accessibles uniquement à l'intérieur de sa propre organisation. Matériels, logiciels et infrastructures sont la propriété de l'organisation qui les contrôle, mais cette configuration fait souvent appel à un hébergement et à une gestion assurés par un tiers.

— **Cloud Hybride** : Les deux approches précédentes sont combinées. Les données considérées comme critiques par l'entreprise sont maintenues dans des domaines privés, alors que les données moins critiques sont logées dans un Cloud public. Entre les deux peut être mis en place une passerelle de façon à permettre l'échange de données de manière sécurisée. La sécurité des données constitue donc un facteur déterminant de prise de décision dans les choix d'architecture Cloud [7].

— **Cloud Communautaire** : Ce type de Cloud est dédié à une communauté spécifique qui intègre plusieurs organisations ayants des interêts communs. L'infrastructure de ce Cloud est partagée par des organisations. Exemples de Clouds d'un domaine spécifique : Les Clouds scientifiques et les Clouds du calcul de haute performance ou HPC. Le premier exemple se rapporte à un Cloud conçu pour fournir des moyens expérimentaux à des projets scientifiques et éducatifs. Les Clouds HPC offrent un soutien adéquat à la communauté HPC [25].

1.2.4 Caractéristiques du Cloud

Le modèle Cloud Computing est généralement défini par les cinq caractéristiques essentielles suivantes [5] :

- **Accès aux services par l'utilisateur à la demande** : La mise en œuvre des systèmes est entièrement automatisée et c'est l'utilisateur, au moyen d'une console de commande, qui met en place et gère ses données à distance.
- **Accès réseau large bande** : Ces centres de traitement sont généralement raccordés directement sur le backbone d'Internet [1] pour bénéficier d'une excellente connectivité. Les grands fournisseurs répartissent les centres de traitement sur la planète pour fournir un accès aux systèmes en moins de 50 millisecondes de n'importe quel endroit.
- **Réservoir de ressources (non localisées)** : La plupart de ces centres comportent des dizaines de milliers de serveurs et de moyens de stockage pour permettre de gérer l'augmentation rapide des charges. Il est souvent possible de choisir une zone géographique pour mettre les données proches des utilisateurs.
- **Facturation à l'usage** : La facturation est calculée en fonction de la durée et de la quantité de ressources utilisées.
- **Redimensionnement rapide (élasticité)** : La mise en ligne d'une nouvelle instance d'un serveur est réalisée en quelques minutes, l'arrêt et le redémarrage en quelques secondes. Toutes ces opérations peuvent s'effectuer automatiquement par des scripts. Ces mécanismes de gestion permettent de bénéficier pleinement de la facturation à l'usage en adaptant la puissance de calcul au trafic instantané.

1.2.5 Inconvénients des Clouds

Nous avons vu tout ce que le Cloud peut offrir, néaumoins, il ne garantit pas la confidentialité et la sécurité des données, car l'hébergement de ces données se fait, en effet, en dehors de l'entreprise, dans un service de base mis en disposition par un fournisseur. Le risque de voir ces données finir en situation de vol ou de mauvaise utilisation demeure donc une possibilité.

De plus, la réputation du Cloud en a pris un coup en avril 2011, lorsque de nombreux clients des services d'Amazon Web Services n'ont pu accéder au Cloud EC2 géré par l'Américain durant quelques heures, mettant de nombreux sites web hors-ligne. Amazon s'est défendu en rappelant que ce genre d'avaries pouvait arriver de temps en temps, comme n'importe quelle avarie matérielle, ici un problème de serveurs dans un de ses centres de

1. Principales voies de données situées entre les grands réseaux stratégiquement interconnectés, ou routeurs de base d'Internet.

traitement de données ou datacenters. Le problème, c'est que cette avarie système aurait entrainé la perte définitive d'un nombre conséquent de données [11].

1.3 Quelque Clouds importants

Avec l'apparition du paradigme Cloud Computing, de nombreuses entreprises et organismes de recherche ont investi des ressources dans la construction de plates-formes de cloud efficaces. Nous allons en définir les principales d'entre elles :

Amazon Web Services : Amazon Web Services (AWS) sont les services de Cloud offerts par Amazon pour leurs clients publics sur Internet. La collection d'Amazon Web Services comprend les services d'informatiques, de stockage et de réseau à plusieurs niveaux (de l'infrastructure aux plates-formes). Traditionnellement, AWS sont de type IaaS. Cependant, ils ont évolué au fil des ans pour inclure les services PaaS. Le service informatique principal est Amazon Elastic Cloud Compute (EC2). EC2 fournit des machines virtuelles munies d'un système d'exploitation préféré, Destinées aux développeurs pour 1) faciliter l'accès aux ressources de Cloud à l'échelle du Web, 2) une capacité de calcul redimensionnable dans le cloud et 3) le déploiement d'une grande variété d'applications avec un contrôle sur l'ensemble de la pile logicielle. En outre, AWS fournit des services de stockage multiples. Le service pour stocker et gérer les données non structurées est service Amazon Simple Storage (S3). S3 permet aux utilisateurs de stocker leurs objets de données dans le Cloud par lot. Afin de soutenir le stockage de données structurées, AWS a introduit le service Amazon DynamoDB. DynamoDB est un service de stockage de base de données NoSQL entièrement géré et qui sert ses clients sur Internet via une interface de programmation applicative (API) [1].

Microsoft Windows Azure : Windows Azure est une plate-forme de services Cloud offerts par Microsoft. Windows Azure offre une vaste gamme de services PaaS pour ses clients, tant au niveau de l'infrastructure qu'au niveau de la plate-forme. Ces services comprennent :

- des plateformes création d'applications Web hautement disponibles et évolutives ;
- des plates-formes pour le déploiement et la gestion à grande échelle ;
- Stockage dans le cloud durable et évolutif ;
- location des ressources de nature IaaS et des machines virtuelles soit Windows ou Linux.
- Gestion de bases de données SQL, etc [12].

Google App Engine : App Engine est la plate-forme de Google qui est considérée comme un service de Cloud. App Engine permet aux clients de créer des applications Web, et fournit des outils qui évoluent automatiquement et élastiquement jusqu'à l'obtention de ressources pour les applications déployées. Les langages de programmation pris en charge de la plate-forme sont Python, Java, et Go. Le stockage et la gestion des données sont orchestré par le Datastore. Datastore est une base de données NoSQL, qui fonctionne en mode High Replication Datastore (HRD). Ce système est tolèrent aux erreurs : les données sont répliquées sur de multiples datacenters pour que l'application puisse continuer à fonctionner même en cas d'évènement catastrophique (ex : un datacenter en panne). Go est un langage de programmation compilé et concurrent inspiré de C et Pascal.Python est un langage de programmation interprété, portable (i.e. fonctionne aussi bien sous Windows que sous Linux ou Mac OS sans aucun changement de code). Java est un langage de programmation orienté objet qui est portable [16].

1.4 Les clusters dans le Cloud

Les services Cloud offrent des clusters polyvalents et prêts pour les applications des organisations, qui ont besoin de capacités de calcul supplémentaires et rapide à moindre coût, afin de supporter leurs traitements applicatifs à hautes performances. Il existe deux catégories de cluster : les clusters physiques et les clusters virtuels. Il y a plusieurs différences et similitudes entre les deux. Un cluster physique est une collection de serveurs (machines physiques) reliés par un réseau physique. Un cluster virtuel est un ensemble de serveurs de données virtuels, parallèles ou distribués, qui sont interconnectés entre eux en utilisant des réseaux virtuels à haut débit.

1.4.1 Les techniques de clustering

Dans cette section, nous allons présenter quelques méthodes de clusterisation, détaillé dans [10] :

L'algorithme de Girvan et Newman

L'algorithme de Girvan et Newman est basée sur la centralité d'intermédiarité (edge-betweeness-centrality). Cette notion est apparue pour la première fois dans le champ des sciences sociales pour déterminer le rôle de chaque acteur (nœud) dans un réseau social

(graphe), le but étant alors de détecter des communautés partageant un intérêt commun. Cette mesure d'intérêt commun est basée sur la notion de plus court chemin dans un graphe. Si deux nœuds sont connectés, il peut exister plusieurs chemins entre eux. Les plus courts chemins sont ceux pour lesquels le nombre de nœuds sur le chemin est minimal. Il peut exister plusieurs plus courts chemins entre deux nœuds, si certains chemins ont la même longueur minimale. Intuitivement, il s'agit de repérer des " points centraux " dans le graphe autour desquels on va opérer des regroupements.

La "betweenness centrality" d'un nœud k est la somme pour toutes les différentes paires de nœuds i et j dans le graphe. La méthode de détection des clusters est un algorithme en deux passes.

- La première passe consiste à calculer la "edge-betweenness centrality" pour chaque arc du graphe, à enlever celle possédant la plus forte, puis à recalculer les "edges-betweenness centrality" pour tous les arcs du graphe résultant jusqu'à ce que tous les arcs aient été enlevés.
- La seconde passe est la construction des clusters à proprement parler. Au début, chaque cluster consiste en un unique nœud. Les arcs entre clusters sont ajoutés dans l'ordre inverse de celui dans lequel ils ont été supprimés dans la première passe. à chaque fois qu'un arc rejoint deux nœuds qui font partie de deux clusters différents, tous les nœuds des deux clusters sont groupés dans un seul cluster. à chaque fois qu'un arc joint deux nœuds qui font partie d'un même cluster, l'ensemble des clusters reste inchangé.

On notera que cette approche récente (2003) est considérée comme l'une des plus performantes à l'heure actuelle et réussit à partitionner des graphes ayant de l'ordre de 100000 arêtes.

La méthode MCL

L'algorithme MCL (Markov Cluster algorithm) est un algorithme de partitionnement rapide et capable de prendre en entrée de très grands graphes. Il est basé sur la simulation de flots stochastiques dans un graphe. Il a été mis au point par Stijn van Dongen au Centre for Mathematics and Computer Science (CWI).

Les clusters naturels dans un graphe sont caractérisés par la présence d'un grand nombre d'arcs entre les membres de ce cluster, et on peut s'attendre à ce que le nombre de chemins de longueur supérieure entre deux nœuds arbitraires dans le cluster soit grand. En particulier, ce nombre devrait être grand, relativement aux paires de nœuds appartenant à des clusters naturels différents. Vu sous un autre angle, une marche aléatoire dans le graphe va peu fréquemment aller d'un cluster naturel à un autre. L'intuition derrière l'algorithme correspond

donc à ce que va faire un être humain pour repérer des regroupements de nœuds : trouver les " gros " blocs de nœuds fortement connectés, groupes qui sont eux-mêmes peu liés entre eux.

L'algorithme MCL trouve la structure en cluster d'un graphe grâce à une procédure mathématique de bootstrapping. Le processus calcule de manière déterministe les probabilités des marches aléatoires dans le graphe, et utilise deux opérateurs transformant un ensemble de probabilités dans un autre. Cela est rendu possible par l'utilisation du langage des matrices stochastiques (aussi appelées matrices de Markov), qui formalise le concept mathématique de marches aléatoires dans un graphe. Informellement, une matrice stochastique donne les probabilités de transitions entre deux nœuds d'un graphe.

L'algorithme MCL simule des marches aléatoires à l'intérieur d'un graphe en alternant deux opérations appelées expansion et inflation.

L'expansion correspond à calculer des marches aléatoires de longueur supérieure, ce qui signifie des marches aléatoires avec beaucoup d'étapes. Il associe des nouvelles probabilités à toutes les paires de nœuds, où un nœud est le point de départ et l'autre est la destination. Puisque les chemins de longueur supérieure sont plus courants à l'intérieur d'un cluster qu'entre des clusters différents, les probabilités associées avec les paires de nœuds appartenant au même cluster vont, en général, être relativement grandes car il y a beaucoup de chemins allant d'un nœud à l'autre. L'inflation va donc avoir pour effet d'augmenter les probabilités de marches intra-cluster et va diminuer les marches inter-cluster. Ceci est obtenu sans connaissance à priori de la structure des clusters. C'est simplement le résultat de la présence d'une structure en cluster.

Finalement, itérer les expansions et inflations résulte en la séparation du graphe en plusieurs segments. Il y a plus de chemins entre ces segments, et la collection des segments résultants est simplement interprétée comme un clustering.

D'après la définition de l'algorithme MCL, il est basé sur un paradigme différent de tous les autres algorithmes basé sur des systèmes de liens. En effet, son fonctionnement est probabiliste, et sa terminaison basée sur une notion de convergence d'un processus aléatoire.

La méthode Graclus

L'algorithme utilise des techniques extrêmement complexes qui nécessitent un très fort background en théorie des graphes. Les paramètres de l'algorithme font qu'il est difficile de l'utiliser sans fixer à l'avance le nombre de clusters que doit contenir le graphe après application de l'algorithme.

L'algorithme se compose de deux phases distinctes : une phase dite de coarsening (littéralement " rendre plus brut ") du graphe, suivie d'une phase de raffinement.

La phase de coarsening se déroule de la manière suivante : Etant donné le graphe initial G0, des graphes vont être construits et être de plus en plus petits G1, G2, ... (c'est-à-dire ayant de moins en moins de nœuds). Pour passer de Gi à Gi+1, des nœuds du premier graphe vont être combinés en " supernoeuds ". Le poids d'une arête (initialement 1) entre deux supernoeuds est la somme des poids des arêtes qui existaient entre les nœuds qui ont conduit aux supernoeuds considérés. L'approche de Graclus consiste, pour cette phase de coarsening, à visiter chaque sommet du graphe. Lorsqu'un sommet x n'a pas été précédemment marqué et que tous ses voisins sont marqués alors il sera marqué à son tour, sinon il sera fusionné avec un voisin non marqué.

La phase de coarsening est itérative sur les graphes successivement crées et s'arrête lorsqu'un graphe qui contient peu de nœuds est construit. Le critère généralement retenu est la construction d'un graphe qui possède moins de 20k nœuds où k est le nombre de clusters souhaités. Une fois cette phase finit commence une phase de raffinement dont le but est de réattribuer les nœuds fusionnés aux clusters crées lors du partitionnement initial : Si un supernoeud est dans un cluster, alors les nœuds qui ont été fusionnés pour créer ce supernoeud sont également dans le cluster. Cet algorithme est peu utilisable, car il faut connaitre à l'avance le nombre de clusters que doit contenir le graphe.

1.4.2 Avantages des clusters dans le Cloud

- Les nœuds de clusters dans le Cloud peuvent être soit des machines physiques ou virtuelles, sachant que plusieurs machines virtuelles avec différents systèmes d'exploitation peuvent être sur le même nœud physique.
- Les clusters travaillent ensemble dans l'exécution de calculs et de tâches de données.
- Les clusters virtuels sont responsables de la répartition de la charge entre les centres de données.
- Les clusters semblent alors être une des meilleures solutions pour l'accès rapide aux données et restent transparents pour les applications clientes.
- Le nombre de nœuds dans un cluster virtuel peut augmenter ou diminuer dynamiquement.
- Les défaillances de nœuds dans un cluster virtuel ne nuit pas aux nœuds physiques hôtes.
- Permet le partage des ressources physiques et un taux d'utilisation plus élevé avec un stockage optimal.

— Le Clustering évite un accès interrompu aux données et aide également lorsque la connectivité réseau ou de stockage est perdue.
— Le Clustering des centres de données qui sont distribués permettent des données hautement disponible pour les clients sans aucun délai [27].

1.5 Les défis des Big Data dans le Cloud

Le taux de croissance remarquable de Big Data présente des défis et des problèmes sans précédent à des experts en informatique. L'un des premiers problème majeur est de nature de l'infrastructure. Les entreprises et les organisations font face à un tsunami de données écrasantes qui dépasse leurs capacités d'infrastructure. L'émergence du paradigme Cloud Computing fourni un filet de sécurité car il offre une quantité infinie de ressources à la demande sur Internet à un prix équitable.

1.5.1 défis posés par les Big Data

Les propriétés citées dans (Section 1.1.1) des Big Data, posent multiples défis, malgrè l'usage de la plateforme Cloud [16] :

— **La durabilité des données** : La durabilité des données est une préoccupation majeure pour les applications Big Data. La perte de données peut avoir diverses conséquences désastreuses dans les applications critiques. Durant des années, les architectures de stockage ont proposé des moyens innovants afin d'assurer la durabilité des données en s'appuiant sur un grand niveau de techniques impliquant la réplication et des mécanismes d'authentification. L'approche la plus populaire au niveau du stockage de disque est celle de RAID (pour Redundant Array of Independent Disks) [2]. Les techniques de RAID combine plusieurs pilotes de disques avec de multiples algorithmes de distribution de données pour assurer la durabilité. Cependant, une technologie pareille n'a pas le niveau des Clouds actuels.
— **Accès rapide** : L'augmentation du volume, variété et de la vélocité des données est associé avec un besoin grandissant pour une performance élevée et un accès rapide. Pour répondre à ce défi, compte tenu du volume important des données, plusieurs techniques sont implémentées pour qu'elles soient utilisées par les fournisseurs Cloud et les entreprises Big Data. Les avancés dans les technologies memoire flash et les disques SSD (pour Solid State Drive) [3] réduisent le temps d'accès de huit fois comparé

2. Assemblage de disques multiples pour créer de gros disques logiques fiables, cela offre une rapidité matérielle et logicielle et une certaine flexibilité, et est caractérisé par sa distribution géographique.
3. Matériel informatique permettant le stockage de données sur de la mémoire flash.

aux pilotes classiques. Au niveau logiciel, les nouvelles architectures de systèmes de stockage s'appuit sur la réplication pour avoir un accès plus rapide aux copies locales de données.

— **La disponibilité** : Nous disons que Les données sont disponibles, si toutes (ou la plupart) des requêtes d'accès aboutissent à un succès. La disponibilité est un besoin critique pour plusieurs applications Big Data. Les services web, comme le e-commerce, necessitent un très grand degrès de disponibilité pour les consommateurs qui doivent toujours avoir une réponse de leurs requêtes afin d'éviter les pertes financières.

— **Tolérance aux pannes** : Les infrasturctures de stockage Cloud sont constituées de materiels basiques, Par conséquent, les defaillances sont toujours connues. L'un des défis importants pour les plateformes et les infrastructures Big Data est de maintenir le système opérationnel dans la présence d'un ou plusieurs composant(s) défaillant(s). La redondance et la réplication sont des moyens connus pour faire face à la tolérance aux fautes.

Conclusion

Nous avons présenté tous ce qui peut caractériser le Cloud, à savoir les modèles Cloud, ses services, ainsi que ses avantages et inconvénients. Et avec les défis posés par les Big Data concernant le traitement et le stockage, le Cloud est l'implémentation la plus appropriée. Dans le chapitre qui suit nous allons traiter deux techniques permettant d'assurer la disponibilité et l'homogénéité des données dans le Cloud, nous parlons ici de la réplication et de la cohérence.

Chapitre 2

Réplication et Cohérence des données

Introduction

Une donnée est une information au niveau du système, qui possède une valeur que l'on peut lire ou modifier ; elle peut être centralisée c'est-à-dire localisée sur un seul site, alors une défaillance de ce site rendrait la donnée inaccessible. La réplication peut être solution à ce problème, toutefois celle-ci soulève la question de la cohérence des données entre les répliques. Nous allons dans ce chapitre présenter la réplication et la cohérence, deux concepts fondamentaux dans notre travail.

Nous allons présenter la réplication et la cohérence dans les bases de données et dans le cache, car elles y sont avant d'être établies dans le Cloud.

2.1 Réplication dans les systèmes de gestion des bases de données

2.1.1 Aperçu sur les bases de données

Une base de données (BDD) est une collection cohérente de données structurées. Une entité de base de données est une paire (nom, valeur) sur laquelle des opérations de lecture et d'écriture peuvent être appliquées. Un système de gestion des bases de données ou SGBD est un ensemble de logiciels permettant de gérer et manipuler de manière efficace une BD. Une base de données est dite cohérente, si elle satisfait un certain nombre de contraintes. Une transaction est une unité de travail définie par Gray en 1980 [26], elle regroupe une suite d'opérations qui doivent être exécutées sur la base des données. Pour être valide, une transaction doit présenter les propriétés "ACID", c'est à dire :

Atomicité : Elle assure qu'en cas de succès, toutes les opérations sont validées et qu'en cas d'échec aucune opération n'est appliquée.

Cohérence : La transaction ne doit pas remettre en cause la cohérence de la donnée.

Isolation Une transaction ne peut pas montrer son effet aux autres transactions avant sa validation. Durant l'exécution de la transaction, les données ne peuvent pas être utilisées par une seconde transaction jusqu'à ce que la première soit terminée.

Durabilité : Une transaction validée ne peut pas être remise en cause. Les conséquences sur la donnée sont persistantes et ne peuvent pas être supprimées.

Remarque L'application d'une transaction valide modifie de ce fait l'état de la donnée qui va être permanent, et aucun incident technique (i.e crash) ne doit pouvoir engendrer une annulation des opérations effectuées durant la transaction.

Une base de données répliquée est composée d'un ensemble de répliques stockées sur différents nœuds d'un SGBD réparti. Dans ce dernier, la mise à jour de la base de données répartie s'exécute soit selon une approche dite **copie-primaire** (la mise à jour est centralisée sur un site puis propagée aux autres sites), ou selon une **approche distribuée** (la mise à jour peut être effectuée sur n'importe quel site). la propagation des mises à jour est gérée par le SGBD reparti et peut être effectuée de manière synchrone ou asynchrone.

— **La propagation synchrone** : Consiste à valider une transaction après l'avoir propagée aux autres répliques. Cette approche assure la cohérence des données et garantit la propriété d'atomicité mais le retardement de la validation des transactions provoque un surcoût des temps de réponse et engendre de ce fait des problèmes de performance et de disponibilité des données.

— **La propagation asynchrone** : Consiste à valider localement la transaction, avant de la propager aux autres répliques. Cette validation locale permet aux utilisateurs de pouvoir continuer à travailler en mode déconnecté. Une validation définitive est alors effectuée à la reconnexion lors d'une étape de synchronisation. Cependant, les opérations effectuées sur le site local pendant la déconnexion peuvent être en conflit avec d'autres opérations qui ont déjà été validées par le SGBD reparti, il faut alors vérifier la cohérence mutuelle des bases de données. Un autre inconvénient de la propagation asynchrone est que la propriété d'atomicité ne peut être garantie, si une panne se produit avant qu'une transaction validée localement puisse être propagée aux autres sites alors cette transaction est perdue.

Les transactions dans les bases de données sont fréquentes et les écritures conflictuelles ne sont pas rares. la cohérence des bases de données est difficile à maintenir et notamment

la résolution des conflits sont souvent laissée à la charge des utilisateurs qui ne sont pas forcément qualifiés pour effectuer cela [31].

2.2 Réplication du cache

La mise en cache est la technique consistant à stocker une copie des données temporairement dans un support de stockage rapidement accessibles dans la CPU. Cette technique peut rencontrer des défis qui comprennent, par exemple, le problème de cache warm-up, où le cache doit être chargé avec suffisamment de données actives pour réduire les échecs du cache et de lui permettre de commencer à améliorer les temps de réponse Entrée / Sortie [4].

— Dans un système typique, plusieurs caches partagent un bus commun, à la mémoire principale.
— Chacune dispose également d'un processeur attaché qui demandent des lectures et écritures. L'objectif collectif des caches est de minimiser l'utilisation de la mémoire principale partagée [2].

Le problème associé à la réplication est la cohérence des données répliquées. En effet, des processeurs peuvent avoir des valeurs différentes pour le même emplacement de mémoire. Pour ceci, MESI (Modified, Exclusive, Shared, Invalid), un protocole de cohérence de cache utilisé dans les systèmes multiprocesseur, est présenté. Pour ce protocole, toute ligne de cache peut être dans l'un des quatre états :

Modified (M) : La ligne de cache a été modifiée et est différente de la valeur correspondante à la mémoire centrale. La ligne en mémoire principale doit être mise à jour avec la version locale avant que d'autres puissent la lire.

Exclusive (E) : La ligne de cache est identique à la valeur correspondante à la mémoire principale et est la seule copie en cache, la cohérence est assurée.

Shared (S) : La ligne de cache est identique à la valeur correspondante à la mémoire principale, mais des copies peuvent exister dans d'autres caches.

Invalid (I) : Indique que cette ligne de cache n'est pas à jour.

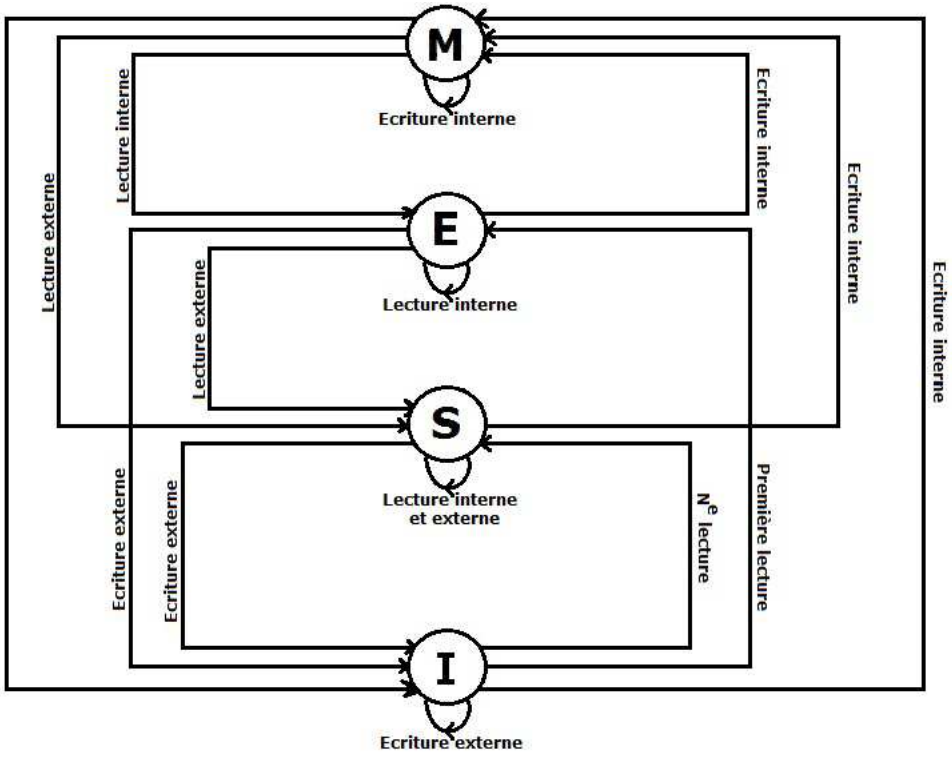

FIGURE 2.1 – Schéma représentant le protocole MESI.

2.3 La réplication des données dans le Cloud

Dans cette section, nous allons détailler un des axes de notre travail, qui est la réplication dans le Cloud.

Définition 2.3.1. La réplication est la copie continue des changements de données d'un site principale vers un site secondaire. Les deux sites sont généralement situés dans différents serveurs physiques, résultant dans le cadre de l'équilibrage de charges en répartissant les requêtes, offrant ainsi une capabilité de reprise, comme dans le cas du Cloud. Le serveur secondaire peut être configuré comme un serveur de secours en cas de défaillance du serveur principal [3].

Remarque Le processus de réplication n'arrête pas le processus de traitement sur les données répliquées.

2.3.1 Avantages de la réplication dans le Cloud

La réplication présente plusieurs avantages :

— Disponibilité de données : permettre à un serveur secondaire de prendre rapidement la place du serveur principal lorsque celui-ci échoue, ou alors lorsque les répliques répondent aux requêtes qui sollicitent une même donnée. S'il y a une perte de toute connectivité réseau, avoir une réplique complète signifie qu'il y a toujours accès à toutes les données.
— Favorisation du partage et du parallélisme ou équilibrage de charges : permettre à plusieurs serveurs de servir les mêmes données.
— Amélioration des performances d'accès aux données répartis (meilleur temps de réponse, diminution du temps de transfert des données, etc) [31].

2.3.2 Techniques de réplication

Avant de concevoir une stratégie de réplication, nous devons d'abord répondre à ces questions [34, 35] :

1. Quoi répliquer (problème de sélection pour la réplication) ?
2. Où placer la réplique (problème d'emplacement de répliques) ?
3. Quand et comment mettre à jour la réplique (problème de mise à jour et de contrôle) ?

Concernant la première interrogation : les ressources distribuées devraient être répliquées selon la popularité, afin de maximiser la probabilité de satisfaction des requêtes utilisateurs (taux de succès des demandes).

En terme de "Où placer la réplique" : Pour raison de sécurité, de coût, de facilité de gestion, de capacité de stockage, etc, certains site ont une copie et d'autres pas. Sur quels sites doit-on alors placer les répliques ?

Nous citons deux critères :

Distance : les répliques sont placées sur les sites qui ont plus de chance de les demander ; Cela leur permet de chercher et de trouver les ressources demandées, et de réduire les délais de recherche et de téléchargement.

Coût : le coût du placement des répliques est la somme de 3 termes :

— Le coût de stockage des répliques.
— Le cout des lectures.
— Le coût des écritures.

Il est facile de voir que la multiplication des répliques réduit le coût des lectures et augmente celui du stockage et des écritures, alors que la diminution du nombre de répliques a l'effet inverse. Néanmoins, la réplication implique une gestion plus complexe des mises à jour et nécessite une vérification de la cohérence entre les données répliquées et la donnée de référence.

Concernant la dernière interrogation : le processus de réplication a lieu lorsqu'une donnée est mise en ligne par un client et en crée au plus une réplique.

Il existe trois méthodes de réplication :

— **réplication instantanée** : Les données sont copiées du site principale vers les secondaire. Les changements au niveaux des sites secondaires doivent provenir du principal. Ainsi, seuls les secondaires sont intérrogés, mais leurs données ne peuvent être modifiées par les clients.

— **Réplication par fusion** : Les données sont combinées de deux ou plusieurs site vers un troisième un site. Cette méthode est plus difficile à mettre en œuvre qu'une réplication instantanée.

— **Réplication transactionnelle** : La donnée est copiée complètement, suivit des mises à jours qui sont copiées periodiquement des sites principaux vers les sites secondaires.

2.3.3 Types de réplication

Réplication synchrone : Garantit que, lorsqu'une transaction met à jour une réplique primaire, toutes ses répliques secondaires sont mises à jour dans la même transaction. L'avantage essentiel de la mise à jour synchrone est de garder toutes les données au dernier niveau de mise à jour. Le système peut alors garantir la fourniture de la dernière version des données quel que soit la réplique accédée. Les inconvénients sont cependant multiples, ce sont d'une part, la nécessité de gérer des transactions multiples coûteuses en ressources et d'autre part, la complexité des algorithmes de gestion de panne d'un site, etc. C'est pour cela que l'on préfère souvent le mode de mise à jour asynchrone.

Réplication asynchrone : Lorsqu'une transaction met à jour une copie primaire, chaque copie secondaire est mise à jour dans une transaction séparée. L'avantage est la possibilité de mettre à jour en temps choisi des données, tout en autorisant l'accès aux versions anciennes avant la mise à niveau. L'inconvénient est bien sûr que l'accès à la dernière version n'est pas garanti [30]. La réplication asynchrone se base sur deux approches illustrées par les figures (Figure 2.2 et Figure 2.3) :

— Dans l'approche basée sur l'état, la réplique source est mise à jour, ensuite, le sous-système transmet l'état sur les répliques par la fusion de l'état livré à l'état local.

— Dans l'approche basée sur les opérations, ici le sous-système envoie l'opération de mise à jour et ses paramètres à toutes les répliques (y compris la source).

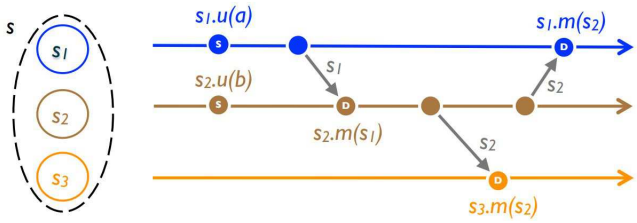

FIGURE 2.2 – Réplication basée sur l'état [38].

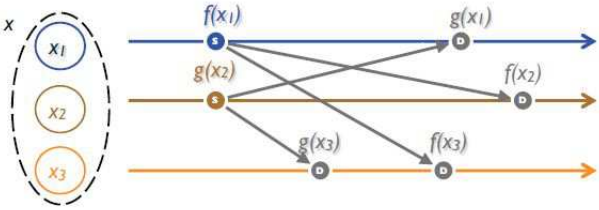

FIGURE 2.3 – Réplication basée sur les opérations [37].

2.3.4 Challenges dans l'environnement de réplication dans le Cloud

Les défis posés pour la réplication dans le Cloud se résument en [37] :

— **Cohérence des données** : Le maintien de l'intégrité et de la cohérence dans l'environnement de réplication est d'une importance majeure. Les applications de haute précision peuvent nécessiter la cohérence stricte des mises à jour faites par transactions.

— **Coût** : La réplication des données sur plusieurs sites engendre des coûts supplémentaires de stockage et d'administration aux fournisseurs cloud. L'idéal est donc minimiser le coût tout en assurant de meilleures performances.

— **Performances** : Les performances d'opérations d'écriture peuvent être médiocre dans des applications qui nécessitent des mises à jour continuelles dans les hauts environnements répliqués. L'idéal est d'adapter le nombre de répliques aux exigeances de l'application, pour réduire la latence des mises à jour et ainsi améliorer ces performances. Autre part

— **Fournir des compromis entre la cohérence, les performance et la disponibilité**.

2.4 La cohérence des données dans le Cloud

Dans cette section, nous allons présenter un des challenges cités dans la Section 2.3.4, et qui constitue un des points essentiels pour notre travail, c'est la cohérence.

Définition 2.4.1. Un stockage distribué cohérent (appelé aussi stockage fortement cohérent) se réfère souvent au fait que chaque mise à jour est appliquée à tous les nœuds concernés en même temps logique.

Une conséquence de cette définition est que, si une donnée est répliquée sur des nœuds différents, elle serait toujours la même sur chaque nœud à chaque instant [18].

2.4.1 Modèles de cohérence

Un modèle de cohérence pour des données réparties spécifie un contrat entre un client et le système de gestion de données (avec engagement mutuel).voici les modèles les plus courant [16] :

Cohérence Forte Dans les systèmes traditionnels de stockage et de bases de données distribuées, la manière instinctive et correcte de traiter la cohérence des répliques était d'assurer un état de cohérence forte de toutes les répliques dans le système à tout moment. La cohérence forte garantie que toutes les répliques sont dans un état cohérent immédiatement après une mise à jour, avant de retourner un succès. Ce modèle de cohérence exige des mécanismes très coûteux en termes de performance et de disponibilité et limitent l'évolutivité du système. Ce ne fut pas un problème dans les premières années de systèmes de stockage distribués les performances nécessaires à l'époque n'étaient pas aussi importantes. Cependant, à l'ère de Big Data et Cloud Computing, ce modèle de cohérence peut être pénalisant en particulier si une telle cohérence forte n'est en réalité pas requise par les applications.

Cohérence faible La mise en œuvre de modèles de cohérence forte impose, dans de nombreux cas, des limitations dans les choix de conception de système et des performances de l'application. Pour surmonter ces limitations, le modèle de cohérence faible a été introduit. L'accès aux données en lecture et en écriture est considéré comme faible s'il remplit les trois conditions suivantes :

- Tous les accès à une variable de synchronisation partagée sont fortement (séquentiellement) commandés. Tous les processeurs perçoivent le même ordre des opérations.
- Aucun accès à une variable de synchronisation n'est fait par un processeur avant que tous les accès aux données précédentes aient été réalisés.
- Aucune lecture ou écriture ne peut être effectuée par un processeur tant qu'un accès à une variable de synchronisation n'est pas terminé.

Cohérence séquentielle : Le modèle de cohérence séquentielle a été défini pour la première fois par Lamport en 1979 [29]. Selon lui un système est cohérent séquentiellement si : `Le résultat de n'importe quelle exécution est le même si les opérations de tous les processus sont exécutées dans un ordre séquentiel, et les opérations de chaque processus`

`individuel apparaissent dans cet séquence dans l'ordre spécifié dans son programme.` A noter que dans un système séquentiellement cohérent, tous les processus doivent s'entendre sur l'ordre des effets observés (On impose un ordre total sur les écritures). Les lectures locales à chaque processus lisent la dernière valeur écrite.

Cohérence causale : Modèle plus faible que la cohérence séquentielle, car on ne considère que des événements reliés par une relation de causalité. Sachant qu'une relation de causalité est définie comme suit : e1 précède causalement e2 noté e1 $\rightarrow$ e2, si :

— e1 et e2 sont sur le même site, et e1 précède e2, ou

— e1 = w(x)a, et e2 = r (x)a (e2 est lecture de la valeur écrite en e1), ou

— e1 $\rightarrow$ e et e $\rightarrow$ e2.

Des écritures causalement liées ($\rightarrow$) doivent être vues par tous les processus dans leur ordre causal. Des écritures causalement indépendantes (parallèles) peuvent être vues dans un ordre différent.

Cohérence FIFO : La cohérence causale peut encore être affaiblie, si on ne considère la causalité qu'à l'intérieur d'un seul processus, non entre processus différents. Dans un processus unique, la causalité se réduit à l'ordre FIFO. Des écritures réalisées par un même processus doivent être vues par tous les processus dans leur ordre de réalisation. Des écritures réalisées par des processus différents peuvent être vues dans un ordre différent

Cohérence cache : Pour chaque emplacement d'une donnée, il existe un ordre total des écritures sur cette donnée. Les écritures sur différents emplacements peuvent être vues dans un ordre différent par différents processeurs .

Cohérence éventuelle : Ce modèle est devenu très populaire dans les dernières années, car il assure une meilleure disponibilité, et des performances plus grandes. On peut tolérer des données incohérentes pendant un certain temps T, si la condition suivante est respectée : Si aucune mise à jour n'a eu lieu pendant T, toutes les répliques deviendront progressivement la dernière version de la valeur. Ce type de cohérence peut poser un problème lorsqu'un client accède successivement à plusieurs répliques différentes d'une donnée. D'où une classe de modèles de cohérence définis à partir de la vue du client

— **Lectures monotones :** Si un processus a lu la valeur d'une donnée, toute lecture ultérieure par ce même processus doit rendre la même valeur ou une valeur plus récente. Ce modèle garantit qu'un client "ne reviendra pas en arrière".

— **Écritures monotones :** Si un processus exécute deux écritures successives sur

une même donné, la deuxième écriture ne peut être réalisée que quand la première a été terminée.

— **Cohérence écriture-lecture :** Si un processus écrit la valeur x' sur une donnée x, toute lecture ultérieure par ce processus doit retourner x'.

— **Cohérence lecture-écriture :** Si un processus lit une donnée x, récupére sa version v, puis écrit sur cette même donnée la valeur x', l'écriture modifiera partout la donnée x ayant une version au moins aussi récente que v.

Conclusion

Après avoir présenté la réplication et la cohérence dans les BDDs et la mémoire cache, et vu quelques types de cohérence et de réplication existants, nous allons, dans le chapitre qui suit, étudier différents protocoles présents dans la littérature, qui gèrent la cohérence des données répliquées dans le Cloud.

Chapitre 3

État de l'art sur la Cohérence et la réplication

Introduction

Ce chapitre contient une présentation sélective des méthodes développées dans la littérature, traitant sur la cohérence des données répliquées dans le Cloud. Nous allons dresser une classification de ces méthodes-là, voir leur mode de fonctionnement et définir leurs avantages et leurs inconvénients.

Dans les solutions analysées certains critères reviennent souvent, nous parlons de : coût, performances, quorums et estampilles. Nous les avons alors classifiés en suivant ces critères-là. Nous obtenons ainsi 4 Classes : La classe Performance, Coût, Estampillage (ou à base d'estampilles), Quorum.

3.1 Solutions basées sur les estampilles

Horodatage pour les jeux massif sur le Cloud

Diao [17] propose une solution de cohérence basée sur les estampilles, dédiée aux jeux massifs en ligne avec des multi-joueurs sur le Cloud. Pour ceci, l'auteur définit plusieurs types de données : données du compte joueur, qui sont stockées sur les bases de données et requièrent une cohérence forte ; données du jeu, qui sont stockées dans le Cloud et sur la machine du client, et qui requièrent un mélange de cohérence éventuelle et de cohérence forte ; données d'état, qui sont stockées sur le Cloud, et le modèle de cohérence approprié est celui proposé par les auteurs. L'auteur définit aussi les serveurs d'accès aux données comme étant responsables de l'échange de données entre la mémoire et le système de stockage Cloud. Une table d'horodatage aussi est définie, elle stocke l'identifiant (ID) ainsi que le

temps de dernière modification (Last Modified Time ou LMT) des données d'état, et l'état de connexion (Log State ou LS). Les auteurs ont adopté une stratégie de réplication. Ils proposent de synchroniser seulement une partie de répliques pendant le jeu pour qu'une mise à jour s'achève sans bloquer celles qui sont ultérieures. Lorsque les joueurs quittent le jeu, toutes les répliques sont mises à jour de façon synchrone, puisque les joueurs en général reprennent le jeu après une période, et donc une lecture ultérieure obtient rapidement les valeurs les plus récentes. Si la requête de lecture provient du moteur de jeu, le nœud ayant la réplique concernée réagit immédiatement. Comme pour les données d'état, une demande d'écriture pour stocker des données est également acceptée par un quorum de répliques au premier abord. Cependant, les valeurs mises à jour doivent ensuite se propager à d'autres répliques de manière asynchrone lorsque le système de stockage Cloud n'est pas occupé, et disposées sous un ordre d'estampilles dans un temps prédéterminé (δ). De cette façon, ce système de stockage Cloud garantit la cohérence des données estampillées du journal. Le processus de lecture et celui d'écriture se déroulent comme suit :

Protocole d'écriture

- La BDD prend périodiquement une image cohérente.
- Chaque serveur d'accès aux données obtient les données d'état correspondantes de l'instantané périodiquement.
- Un serveur d'accès aux données génère un message comprenant l'ensemble de ses données d'état ainsi qu'une nouvelle estampille (TS), puis l'envoie au système de stockage Cloud.
- Dans le Cloud, ce message est divisé en fonction de l'ID de données d'état en plusieurs sous-messages, dont chacun comprend la même TS. De cette façon, l'échec de la mise à jour d'une donnée d'état ne va pas bloquer des mises à jour d'autres données d'état.
- Ensuite, ces messages sont acheminés vers les nœuds appropriés.
- Quand un nœud reçoit un message, il écrit des changements immédiatement dans le journal, met à jour les données, et enregistre les TS en tant que version de l'ID.
- Si une mise à jour est réussie et TS est supérieure à la valeur de LMT de ces données d'état, le serveur d'accès aux données remplace LMT par TS.
- Si un joueur quitte le jeu et les données d'état du client ont été sauvegardés dans le système de stockage Cloud, le LS du client doit être modifié de "login" à "logout", et les données d'état pertinentes dans la BDD doivent être supprimées.

Protocole de lecture

- Quand un joueur commence le jeu, un serveur d'accès aux données obtient d'abord LS à partir de la table d'horodatage.
  - Si la valeur est "login", cela signifie que l'opération de stockage précédente n'est pas complétée et les données d'état sont toujours stockées dans la BDD. Dans ce cas, le joueur obtient la date de l'état de la BDD directement, et le serveur d'accès aux données doit générer une nouvelle TS pour remplacer LMT des données pertinentes d'état ;
  - Si la valeur est "logout", le serveur d'accès aux données obtient alors le LMT, et l'envoie avec une demande de lecture vers le système de stockage Cloud.
- Lorsque le nœud concerné reçoit la demande, il compare le LMT avec sa version de l'ID local.
  - Si elles correspondent, ce nœud répond à la demande de lecture immédiatement.
  - Sinon, la demande de lecture sera envoyée aux autres nœuds.
- Lorsque le serveur d'accès aux données reçoit les données d'état, il les envoie à la BDD, ainsi qu'aux clients appropriés, et modifie les LS de "logout" à "login" dans la table d'horodatage.
- Les données d'état peuvent également être lues par l'application (dans ce cas c'est le moteur du jeu) à des fins statistique. Donc, les données mises à jour ne sont pas nécessaires, de sorte que la comparaison entre LMT et la version de l'ID ne devrait pas y être.

Cette solution permet d'assurer la cohérence par ordre d'estampilles, offrant ainsi de bonnes performances et un bon rendement du système de stockage Cloud. Néanmoins, les horloges doivent être synchronisées, ce qui est un problème majeur pour les systèmes distribués, y compris le système de Cloud. De plus, l'auteur affirme qu'une lecture peut retourner une valeur périmé, puisqu'il n'y aurait pas de propagation tant que le Cloud est occupé. "Si la requête de lecture provient du moteur de jeu, le site ayant la réplique concernée réagit immédiatement ", l'auteur ne détaille pas ici comment elle réagit, ni pourquoi elle réagit spécialement pour le moteur de jeu.

PPS

Pour le maintien de la cohérence des données et le suivi des changements de documents partagés dans le scénario d'édition collaborative, Qinyi et al. [33] proposent une nouvelle structure de données, " la séquence partielle persistante (PPS) ", elle crée des identifiants de position uniques de caractères appelés " estampilles de position " pour associer des méta-

informations et le suivi de leurs modifications. Elle est dite séquence persistante partielle parce que toutes les révisions sont accessibles mais seulement la version la plus récente peut être mise à jour. PPS considère deux propriétés pour la cohérence des données : cohérence éventuelle et la préservation de l'intention, en effet cette structure garantit la cohérence éventuelle d'un document partagé, car elle crée un ordre total pour tous les caractères. Elle satisfait également la préservation de l'intention parce que l'ordre relatif des caractères est capturé par des estampilles de position. Les interactions des éditeurs en collaboration doivent être coordonnées en raison de conflits lors de modifications simultanées. Le PPS est un " commutative replicated data type (CRDT) " qui a été prouvé être commutative pour toutes les opérations considérées être simultanées. *Répliques de tout CRDT garantissent la convergence à la même valeur, malgré un certain nombre de défaillances* [32].

Protocole d'écriture

1. Étant donné s_i et s_j deux estampilles de position, si m caractères $c_1c_2...c_m$ sont insérés entre eux (entre s_i et s_j), ils seront répartis uniformément sur l'espace $d_{ij} = s_j - s_i$ sous la condition $d_{ij} \geq (m + 1)$. Ainsi, les m caractères auront respectivement les estampilles de position s_i+gap, ..., s_i+m*gap, où gap = d_{ij}/(m+1).
2. Au lieu de maintenir chacun d'eux individuellement, ils sont représentés dans un enregistrement compact < s_i+gap, gap, m>, où : s_i+**gap** est l'estampille de position du caractère le plus à gauche, **gap** est la distance entre les caractères consécutifs, et **m** la longueur de la séquence de caractères.
3. Une seule entrée est donc insérée avec la clé s_i+gap et sa valeur est $c_1c_2...c_m$.
4. Si les anciens enregistrements sont mis à jour, ils seront divisés en sous-enregistrements.

Remarque : Lors d'une suppression d'un caractère, son estampille sera affectée pour un autre caractère récemment inséré.

L'avantage est que cette représentation compacte permet d'économiser l'espace disque et d'accélérer la mise à jour et les performances de récupération. Néanmoins, il sera difficile de gérer et les caractères et les estampilles de positions dans le cas des Big data. Aussi dans le cas d'une suppression d'un caractère et d'absence d'écriture, l'estampille de position concernée reste inoccupée.

3.2 Solutions à quorum

Quorums circulaires

Kumar et Agarwal [20] proposent des protocoles de requête (lecture) et de mise à jour (écriture), sous des systèmes à quorum circulaire. Un quorum circulaire est un sous ensemble de serveurs de répliques. Le quorum circulaire est définit comme suit : Les serveurs sont divisés en plusieurs quorums de lecture et d'écriture tels qu'ils forment une structure circulaire. Sur cette structure, il y a des quorums de lecture, d'écriture et autres, et une collection de ces quorums de lecture et d'écriture est construite en respectant les propriétés illustrées dans les équations (equa 3.1, 3.2) :

$$\forall i, j : l_i \in L, e_j \in E, l_i \bigcap e_j \neq \varnothing \tag{3.1}$$

$$\forall i, j : e_i \in E, e_j \in E, e_i \bigcap e_j \neq \varnothing \tag{3.2}$$

Tels que : e_i, e_j sont des quorums d'écriture, **E** est l'ensemble des quorums d'écriture, l_i est un quorum de lecture, **L** est l'ensemble des quorums de lecture. La disposition des quorums (nœuds représentés sur la structure) peut désigner une topologie particulière (diamond, grid, etc.), ainsi illustré dans la figure (Fig. 3.1). Les auteurs ont proposé des systèmes à quorum circulaires α et β, plus appropriés pour la réplication de données dans le scénario de lecture dominante (i.e il y a plus de lectures que d'écritures).

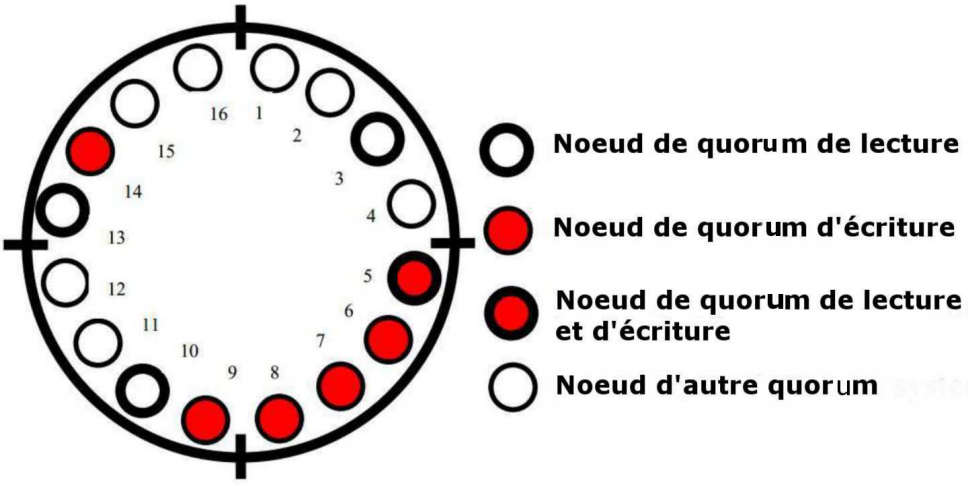

FIGURE 3.1 – Exemple de système de qurom circulaire [20].

Protocole d'écriture

1. Le coordinateur diffuse la transaction T(Ci,i), l'estampille (Ci,i), lecture-set L(T(Ci,i)), écriture-set E(T(Ci,i)), l'adresse du client correspondant et l'ensemble des quorums d'écriture à tous les sites de répliques d'un quorum d'écriture actif choisi dans un message.
2. Lors de la réception de ce message, un site met l'estampille (Ci,i) dans toutes ses files d'attente prioritaires où il existe une estampille (Cj, j), telle que T(Ci,i) correspondant à (Ci,i) et T(Cj,j) correspondant à (Cj,j) sont en conflit. Si la transaction T(Ci,i) n'est pas en conflit avec aucune transaction, elle est insérée dans une nouvelle file d'attente prioritaire.
3. La plus petite estampille dans une file d'attente prioritaire a la plus haute priorité. Par ailleurs, le site envoie une réponse à T(Ci, i) au coordinateur de transactions si deux conditions sont vérifiées.
   — (Ci, i) a la plus haute priorité dans toutes les files d'attente prioritaires où elle existe, et
   — Aucune transaction T(Cj,j), qui est différente de T(Ci,i) et est en conflit avec T(Ci,i), n'est en attente d'une réponse. Si un site attend la réponse de T(Cj,j) qui n'est pas de plus haute priorité, il envoie le message " annuler T(Cj,j) " au coordinateur de transactions.
4. Sur réception d'un message " annuler T(Cj,j) " , le coordinateur renvoi la confirmation de ce message sous forme de " confirme-annuler T(Cj,j) ", s'il n'a pas reçu une réponse de T(Cj,j). Sur réception de " confirme-annuler T(Cj,j) ", un site annule le statut de réponse pour la transaction T(Cj,j).
5. En outre, si le coordinateur de transaction soit reçoit une réponse de T(Ci,i) ou détecte une défaillance d'un site parmi les participants à la transaction, il diffuse un message de supervision (T (Ci, i)) à tous les participants à la transaction. Après avoir reçu le message (T (Ci, i)), le site met ce message dans sa file d'attente prioritaire avec une priorité moindre, et comme traitement de ce message, il en supprime l'estampille (Ci, i). En outre, il exécute toutes les opérations dans le même ordre que celle présente dans la file d'attente. Enfin, il fournit les données requises et leurs numéros de version à l'opération de mise à jour pour l'espace de travail pour un traitement ultérieur de la transaction. Une fois que l'exécution d'une opération de mise à jour T (Ci, i) complète, le coordonnateur valide la transaction.

Protocole de lecture

1. Un coordinateur des transactions i diffuse " lecture-set " L(T(Ci,i)) sur les sites de répliques d'un quorum de lecture choisi.
2. Si réception de lecture-set, ou si le site est le coordinateur de transactions, il traite cette demande, tri ses données selon le numéro de version, et renvoi la donnée ayant la plus récente valeur.
3. Si le coordinateur de transaction ne reçoit pas une réponse de tous les sites du quorum de lecture choisi après un certain délai d'attente, le coordinateur envoi un message d'échec à tout le système et au client. A la réception de l'échec, tous les sites de répliques terminent l'exécution de toute des transactions.
4. Dans le cas d'un échec, le client réémet sa demande sous forme de transaction à n'importe quel site de réplique actif. Le coordonnateur des transactions exécute une opération de mise à jour en utilisant le protocole de mise à jour.

Malgré l'amélioration des performances que cette solution offre, surtout dans le scénario de lecture dominante, les auteurs n'ont pas expliqué quelle est la procédure du choix d'un quorum. De plus, dans le cas d'un échec d'une requête, le client réémet sa demande de lecture, et le coordonnateur lance une mise à jour, dont les auteurs n'ont pas mentionné avec quel valeur. Mais encore, " sur réception d'un message " annuler T(Cj,j) " , le coordinateur renvoi la confirmation de ce message sous forme de " confirme-annuler T(Cj,j) ", s'il n'a pas reçu une réponse de T(Cj,j) ", les auteurs n'ont pas présenté ce qui se passerait lorsque le coordonnateur reçoit un message " annuler T(Cj,j) " après avoir reçu une réponse à T(Cj,j).

Protocol LibRe

Kumar et al. [28] proposent un protocole nommé "Library for Replication" (LibRe), qui assure la cohérence en communiquant avec un nombre minimal de nœuds de répliques. L'idée est de réaliser des lectures cohérentes à l'aide d'un Registre d'information qui sera distribué à l'intérieur du système de stockage afin d'obtenir de meilleures performances. Le registre utilisé enregistre la liste des nœuds de répliques contenant la version la plus récente des éléments de données nécessaires. Ainsi, en se référant au Registre pendant la lecture, cela permettra une transmission des requêtes de lecture à un nœud de répliques maintenant la version la plus récente de l'élément de données nécessaire.

Le protocole recueille des informations sur les écritures jusqu'à ce qu'elles soient pleinement propagées à tous les nœuds de répliques du cluster. Si une mise à jour ne se propage pas à tous les nœuds, une entrée pour cet élément de données est ajouté à une structure de

données en mémoire appelé Registre. Le Registre est une structure de données clé-valeur en mémoire qui prend l'identifiant des données comme la clé et la liste des nœuds de répliques (adresses IP) qui maintiennent la version la plus récente. Au cours de l'opération d'écriture, chaque nœud de répliques tente d'ajouter son identifiant dans la liste. Si le nombre des identifiants dans la liste atteint le nombre total des nœuds de répliques pour l'élément de données, et en confirmant la convergence de toutes les répliques, alors l'entrée de l'élément de données dans le Registre peut être retirée en toute sécurité.

Soit R_i un ensemble de répliques pour un élément de données d_i, de telle sorte que $Ri = \{r_1, r_2, ...r_n\}$, ou r_x est un identificateur de nœud, et n est le nombre de nœuds des répliques. Ainsi, le premier nœud de répliques disponible à Ri, obtenu via une table de hachage distribuée (THD), tiendra le Registre et disposera d'un gestionnaire de disponibilité et d'un manager de publicité. Ce nœud sera nommé "nœud Registre". Le registre sera réparti sur tous les nœuds du cluster, de façon à ce que chaque nœud dispose d'une entrée de données. Le protocole LibRe est basé sur deux types de messages : message de publicité et message de disponibilité :

Message de publicité : C'est un message qui est envoyé après une opération d'écriture réussie, du nœud de répliques sollicité par le client (ce nœud devient coordonnateur) vers le nœud Registre de manière asynchrone. Le message publicitaire est constitué de la clé de la donnée, la version-id et l'identifiant du nœud d'origine.

Message de disponibilité : Lorsque le nœud coordonnateur reçoit une demande de lecture, il envoie un message de disponibilité au nœud Registre de ces données particulières. Le message de disponibilité contient le message d'origine de lecture envoyé par le client et la clé de la donnée.

Protocole d'écriture

Lorsqu'un nœud de répliques envoie un message publicitaire concernant une mise à jour, le manager de disponibilité du nœud Registre prend les mesures suivantes : Premièrement le protocole LibRe vérifie si la clé de données existe déjà dans le Registre (répliques sont dans un état de divergence), la version-id enregistrée dans le Registre pour la clé de données correspondante est comparée à celle qui est envoyée avec le message de mise à jour. Si la version-id enregistré dans le Registre correspond à la version-id de l'opération, alors l'identifiant de nœud (adresse IP) sera ajouté à la liste des répliques existantes. Si le nombre de répliques dans la liste est le même que le nombre total de répliques pour un élément de données k, alors les répliques sont en état de convergence. Si l'entrée n'existe pas dans le Registre ou la version-id de l'opération est supérieure à la version-id existant dans le Registre : l'entrée est

créé ou réinitialisé avec la version-id de l'opération et l'identifiant du nœud émetteur. Cette configuration permet d'atteindre la politique de " Last Writer Wins " [39].

Protocole de lecture

le Registre conserve des informations sur les nœuds de répliques tenant la version la plus récente d'une donnée k, ces informations seront récupérées du Registre. Si une entrée pour la clé de données existe dans le Registre, l'un des nœuds de répliques de l'entrée sera choisi comme le nœud cible. La méthode `premier()` renvoie le nœud de répliques le plus proche triés par proximité. Si le Registre ne contient pas d'entrée pour les données clés nécessaires, alors, l'un des nœuds de répliques qui sont chargés de stocker l'élément de données seront récupérées localement via DHT (table de hachage distribuée). Enfin, le message de lecture sera transmis au nœud cible choisi.

L'avantage du protocole LibRe est le maintien du Registre. Cependant dans le cas où un nœud rejoint ou sort du cluster, la technique de hachage supporte une redistribution uniforme minimale des nœuds clés. Dans ce cas, il y aura un changement dans le premier nœud de répliques disponible (nœud Registre) pour quelques éléments de données et les informations du Registre pour ces données-clés ne seraient pas disponibles. Dans ce cas, le Registre sera reconstruit sur le nouveau nœud Registre en envoyant des messages publicitaires successifs à ce nœud. Cela peut conduire à de petites incohérences limitées dans le temps dans le système. Par conséquent, LibRe sacrifie la cohérence en faveur de la disponibilité.

3.3 Solutions basées sur le coût

C3

Fetai et al. [22] présentent une nouvelle approche, nommée cost-based concurrency control (C3), qui permet d'adapter les niveaux de cohérence des transactions lors de leur exécution. C3 a été implémenté sur un environnement Cloud comme étant "Data-as-a-Service" (DaaS) et prend en considération tous les coûts qui apparaissent lors de l'exécution. Ces coûts sont déterminés par l'infrastructure pour l'exécution des transactions sur un certain niveau de cohérence (dit coût de cohérence) et, optionnellement, par autres coûts spécifiques aux applications pour la compensation des effets d'accès aux données incohérentes (dit coût d'incohérence). C3 prend en charge des transactions s'exécutant simultanément avec différents niveaux de cohérence, et en même temps, il assure les garanties de chacun de ces protocoles de cohérence. L'architecture de C3 est basée sur les protocoles de contrôle de coût (Concur-

rency Control Protocol ou CCPs), qui sont implémentés en tant que fonctionnalité et modèle de coût. Le Framework de C3 peut être facilement étendu pour prendre en compte d'autres CCPs, simplement en introduisant les composantes CCP correspondants dans le Framework. De plus, C3 offre la possibilité d'activer ou désactiver des composants spécifiques. L'évaluation de C3, faite par les auteurs, montre que les transactions avec le comportement adaptatif surpassent les transactions pour lesquelles le comportement est fixé, non seulement du point de vue des coûts, mais aussi du point de vue des performances. Toutefois, Les auteurs ne présentent pas le modèle DaaS (Data as a Service), sur lequel ce base cette approche. En outre, il est difficile d'imposer aux fournisseurs certains critères qui vont avec les coûts déterminé par C3. Par exemple, si le coût minimal d'un service offert est jugé trop élevé par C3, tous les fournisseurs, dans ce cas, ne peuvent offrir un service qui va avec un coût moindre que le minimum. Et enfin, la solution C3 ne concerne que les transactions, les données peuvent être incohérentes.

Etat de réplique

Pour la cohérence de données et un système de stockage Cloud distribué à faible coût, Aye [13] propose un modèle analytique de file d'attente MM1 sur un Cloud privé afin d'évaluer la probabilité de rejet en fonction des demandes de mise à jour pour gérer les conflits des mises à jour en effet une mise à jour ne peut être écartée que par cette probabilité. Lorsque le nombre de demandes de mise à jour augmente, la probabilité de rejet augmente également de manière significative pour le maintien de la cohérence.

De plus, une méthode est également proposée pour la lecture d'une réplique cohérente de système de stockage cloud privé. Après la mise à jour des données, les répliques seront représentées dans le système avec de multiples versions, certaines d'entre elles seront actives et d'autres seront inactives (nœud défaillant) ; quelques répliques seront mises à jour (à savoir cohérente), certaines seront incohérentes. Dans les résultats de calculs de la probabilistes sur le stockage cloud, une réplique peut transiter entre l'un des quatre états possibles S (0,0), S (1,0), S (1,1) et S (0,1) correspondant à (Inactive, incohérente), (active, incohérente), (active, cohérente) et (inactive, cohérente). Une version cohérente d'une donnée sera lue par le système de stockage cloud seulement si au moins une réplique cohérente est disponible en d'autre termes, il est probable que la réplique active-cohérente dont la probabilité est de P (1,1) soit disponible, le système ne sera donc pas disponible dans les versions P (0,0), P (0,1) et P (1,0).

L'avantage de cette méthode est qu'avec les probabilités de lecture qui changent avec l'augmentation du nombre de répliques, le système finira par contenir au moins une réplique

cohérente, cependant elle ne garantit pas la disponibilité puisque dans le cas de S(0.1) , même quand la réplique est cohérente elle ne peut pas être utilisée car elle est inactive.

3.4 Solutions basées sur les performances

Harmony

Chihoub et al. [15] ont proposé " harmony ", une solution pour gérer dynamiquement la cohérence de données dans le Cloud, en choisissant le niveau de cohérence le plus approprié lors de l'exécution. Cette proposition est dotée d'un modèle d'estimation de taux de lectures périmées (les lectures de données qui ne sont pas à jour) qui est basée sur des calculs probabilistes en considèrent l'état du système de stockage ainsi que les taux d'accès aux données. En se basant sur cette estimation, Harmony augmente ou diminue le nombre de répliques impliquées en ces lectures afin de préserver un petit nombre de lectures périmées tolérable par l'application. Le niveau de cohérence par défaut est le modèle de cohérence éventuelle qui permet la lecture d'une seule réplique. Quand un tel niveau ne peut pas satisfaire les exigences de cohérence d'une application en raison du nombre croissant de lectures périmées, le nombre de répliques cohérentes Xn qui devrait être impliquées dans les requêtes de lecture est calculée afin de fournir un taux de lecture périmée inférieur ou égal au taux de lecture périmée tolérée par l'application. Toutes les requêtes de lecture suivantes seront effectuées avec le niveau de cohérence basé sur Xn.

La lecture peut être périmée si son temps de démarrage Xr est dans l'intervalle de temps entre l'heure de début de la dernière écriture Xw et la fin du temps de propagation des données vers les autres répliques. Voir la figure (Fig. 3.2)

Harmony fournit un compromis entre la cohérence et la performance, en effet elle offre des performances acceptables tout en répondant aux exigences de cohérence de l'application et cela en modifiant dynamiquement Xn selon le taux de lecture périmée. Et elle ne considère pas seulement les exigences de l'application, mais aussi l'état du système de stockage Cloud et le nombre d'accès aux données, des métriques que chaque thread recueille à partir d'un ensemble de nœuds du système. Cependant Harmony ne montre pas comment choisir les répliques cohérentes impliquées et elle n'est appliqué que pour des applications dont les exigences de cohérence sont prédéfinies.

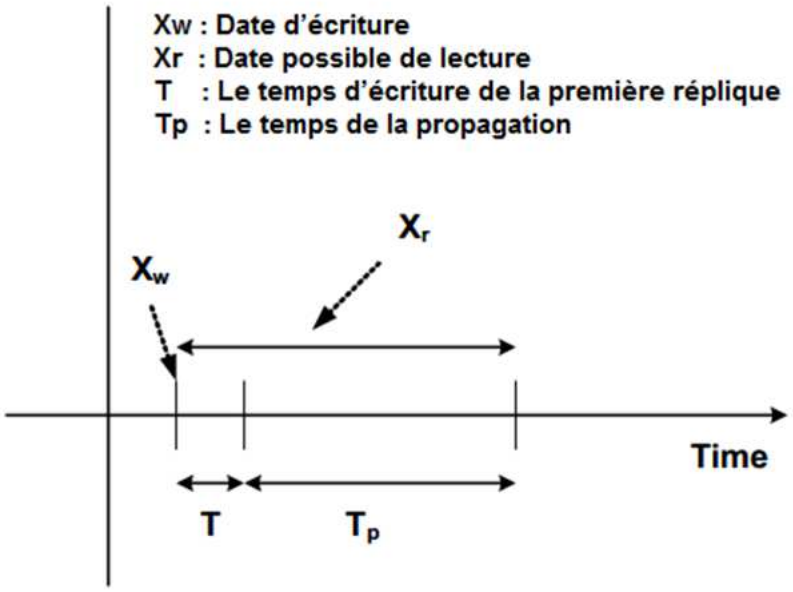

FIGURE 3.2 – situation qui conduit à une lecture périmée [15].

DCaaS

Elgedawy [19] propose un nouveau service de plate-forme middleware pour la gestion de la cohérence, nommé DCaaS. Pour accéder aux données, les développeurs doivent écrire des invocations pour les opérations du service DCaaS. L'auteur propose aussi une approche multi-niveaux de cohérence. Les exigeances en matière de cohérence sont définies dans un plan de cohérence des données indiquant le niveau de cohérence requis pour chaque donnée. L'auteur propose enfin une approche basée sur le quota, qu'utilise DCaaS pour le traitement des données, elle consiste à attribuer un quota de capacités de donnée à chaque Cloudlet (l'auteur désigne par Cloudlet un datacenter). Un Cloudlet peut emprunter des quotas autres que celui qui lui été attribué en envoyant des demandes aux autres Cloudlet, et ces demandes peuvent éventuellement échouer.

DCaaS assure la portabilité du service Cloud, car il découple le niveau SaaS du PaaS, donc, au lieu de solliciter un service SaaS, on instancie un service DCaaS qui utilisera des services PaaS. Le seul changement nécessaire est au niveau de l'interface entre le DCaaS et le PaaS, qui pourrait être manipulé facilement en utilisant des adaptateurs de service. De plus, les expériences faites par l'auteur montrent que l'approche à base de quotas proposée réalisé par le service DCaaS offre un bien meilleur temps de réponse par rapport aux techniques de verrouillage et de blocage.

Une instance DCaaS est située entre deux instances de SaaS et de PaaS, elle assure

donc la cohérence au niveau applicatif. Néanmoins, le niveau de plateforme (PaaS) n'est pas concerné, de même pour le niveau infrastructure (IaaS), et donc, le stockage par exemple peut ne pas être cohérent. Par ailleurs, l'auteur présente dans son article un algorithme d'accès en écriture sous le niveau DCaaS, l'accès en lecture n'a pas sa part. Enfin, l'auteur ne spécifie pas comment une demande d'emprunt échoue.

Conclusion

La réalisation de ce chapitre nous a permis de faire une analyse des différents travaux se rapportant à notre thème. Nous allons dans le chapitre qui suit présenter notre propre travail essayant ainsi de gérer la cohérence des répliques dans le Cloud.

Chapitre 4

Proposition pour la cohérence des données répliquées dans le Cloud

Introduction

Nous avons consacré ce chapitre à la présentation de notre solution. Nous allons définir l'architecture de notre Cloud, la structuration des données, la manière dont le stockage est fait et puis présenté les protocoles d'écriture et de lecture que nous avons proposé afin d'assurer la cohérence des répliques et satisfaire ainsi les requêtes des clients.

4.1 Problématique

Avant de détailler notre proposition, nous allons dans cette section présenter la problématique de notre thème.

Big Data pose plusieurs défis, ainsi mentionné dans la section 1.5. La solution la plus répendue est la réplication. Le problème qui se pose ici c'est la cohérence des répliques, donc si les répliques secondaires ont la même version que la réplique originale. Plus précisément, est-ce que toutes les répliques détiennent la version la plus récente. C'est ce que nous allons traiter dans notre proposition.

4.2 Architecture du Cloud

L'architecture du Cloud est composée de clusters disjoints de datacenters (Fig. 4.1, Fig. 4.2). Chaque cluster est supervisé par un clusterhead (CH), et est composé des quorums de lecture et des quorums d'écriture.

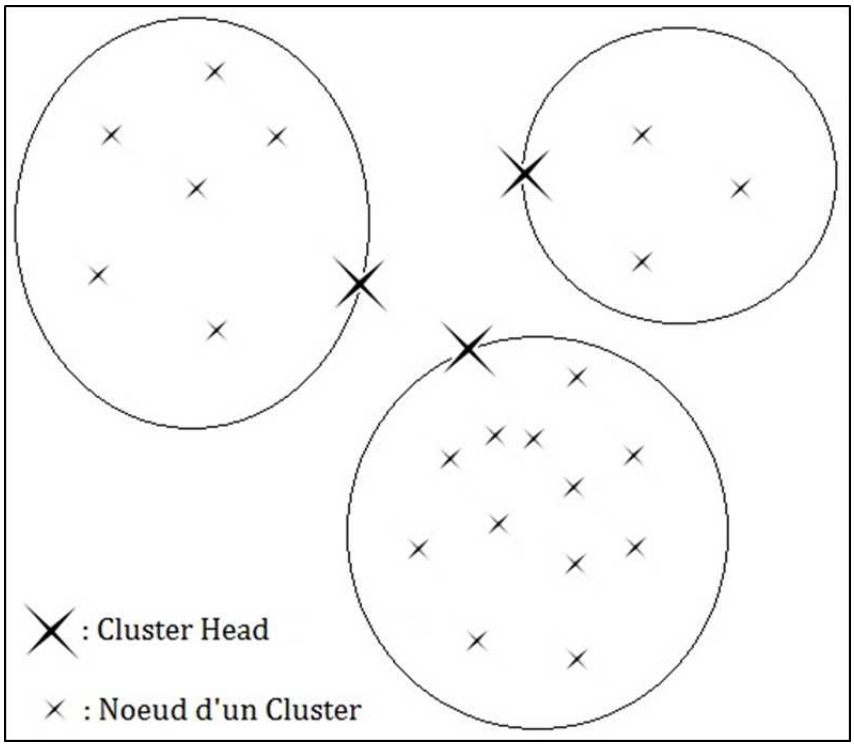

FIGURE 4.1 – Architecture du Cloud proposée.

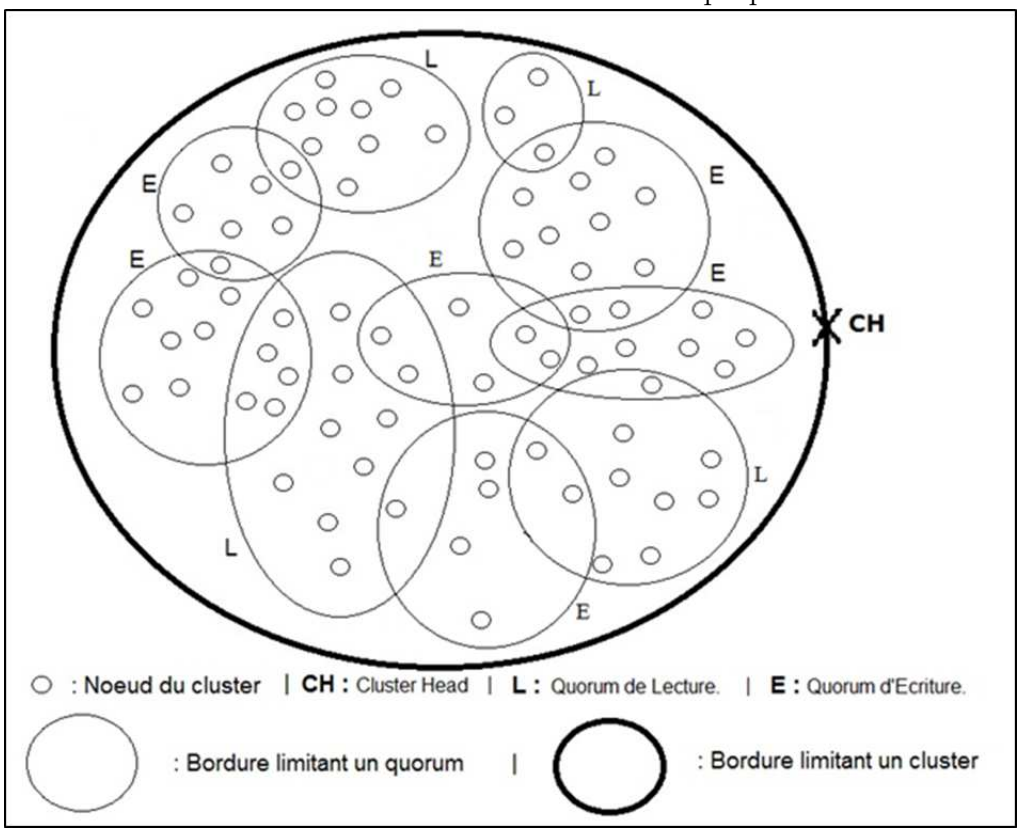

FIGURE 4.2 – Zoom sur un cluster.

4.3 Structuration des données

Au lieu de traiter les données une à une, nous proposons de les regrouper sous une matrice dynamique de dimensions i*c, tel que **i** représente le nombre de lignes qui croit suivant les requêtes de stockage, et **c** le nombre de colonnes qui reste fixe (mais dont la taille est dynamique). Chaque cellule de cette matrice est un vecteur de trois éléments, tels que : le premier concerne l'identifiant de la donnée, le deuxième est sa valeur, et la troisième son estampille (ou sa date). La matrice se remplie ligne par ligne, lors d'une requête de stockage. A noter que CH a les coordonnées (ligne, colonne) de la dernière cellule utilisée, qu'il met à jour à chaque nouvelle requête de stockage. Nous proposons une réplication asynchrone des colonnes de données. Pour chaque quorum, toutes les données existantes sont au niveau de ses nœuds. Pour chaque colonne, nous avons une copie originale et r répliques secondaires. Chaque nœud contient une seule colonne (copie originale ou réplique).

Nous supposons que la phase de distribution des colonnes est achevée, ainsi que la phase de réplication. c et r sont estimés selon le système d'équations suivant :

$$\begin{cases} N = c * (r + 1) \\ (|c - r| = 1) \vee (c = r) \end{cases} \tag{4.1}$$

Tel que **N** est le nombre de nœuds dans un cluster.

	Colonne1			**Colonne2**			**Colonne3**		
	id	valeur	date	id	valeur	date	id	valeur	date
Ligne1	a	23	5161100	b	'chaine'	5161101	c	True	4161600
Ligne2	d	'r'	6151430						
Ligne3									

TABLE 4.1 – Exemple de structure matricielle proposée

4.4 Construction des quorums

Les quorums sont construit de façon statique, tel que :

— L'intersection entre un quorum de lecture et un quorum d'écriture n'est jamais vide et il existe exactement un nœud par intersection.
— Dans chaque intersection, il y a un nœud spécial appelé coordonnateur de quorum.
— Chaque quorum de lecture ou d'écriture est muni d'un coordonnateur.
— Chaque quorum de lecture ou d'écriture contient toutes les données au moins une fois.
— Le nombre de quorums q dans un cluster est de : q=Max(c,r)+1, et :

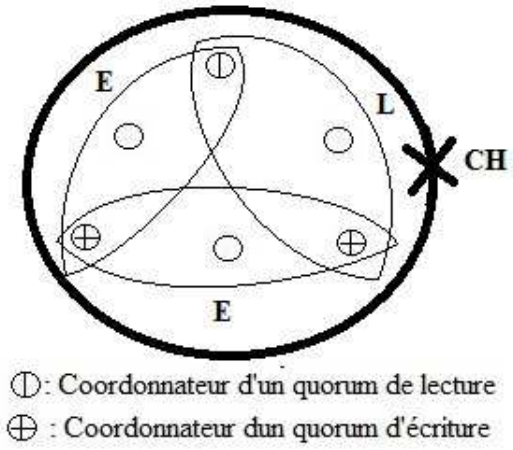

FIGURE 4.3 – Exemple de construction de quorums pour E > L.

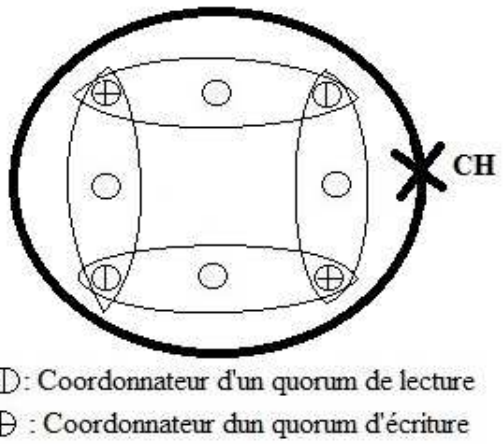

FIGURE 4.4 – Exemple de construction de quorums pour E = L.

1. Si $(c \leq r)$, alors nous avons pour chaque quorum, un nombre (c+1) de nœuds.
2. Si $(c > r)$, alors nous avons pour chaque quorum, un nombre c de nœuds pour (q-1) quorums, et (c+1) pour le dernier quorum. (voir l'exemple dans Fig. 4.5)

Deux cas de figure, illustrés par les figures (Fig. 4.3 et Fig. 4.4), peuvent se présenter dans la construction des quorums de notre cluster :

— **Cas 1** : le nombre de quorums d'écriture E supérieur au nombre de quorums de lecture L.

— **Cas 2** : le nombre de quorums d'écriture est égale au celui des quorums de lecture.

Remarques :

— Chaque nœud d'un quorum connait son coordonnateur.

— Chaque coordonnateur connait les nœuds (les ids) de son quorum.

— Chaque CH connait tous les coordonnateurs de son cluster.

4.4.1 Types de messages

QUERY-IF :<NumDeMsg> Ce message représente la requête de lecture, il contient le nom de la donnée sollicitée, ainsi que le numéro de série.

INFORM :<NumDeMsg> Ce message représente la requête de lecture et est echangé entre un coordonnateur et un CH.

AGREE :<NumDeMsg> Ce message représente la confirmation qu'il n y a pas d'écriture en cours, il contient le numéro de série de la RL.

PROPAGATE :<NumDeMsg> Ce message représente la propagation de la requête encapsulée.

PROPOSE :<NumDeMsg> Ce message représente la donnée envoyée comme réponse à la requête de lecture.

REJECT-PROPOSAL :<NumDeMsg> Ce message est envoyé lors d'une lecture, si la donnée n'existe pas au niveau du nœud.

REQUEST :<NumDeMsg> Ce message représente la RE.

DISCONFIRM :<NumDeMsg> Ce message représente la non-confirmation de la RE.

CONFIRM :<NumDeMsg> Ce message représente la confirmation de la RE.

4.5 Protocole de stockage

1. Lorsqu'un client introduit une nouvelle donnée a, il sollicite le nœud le plus proche, en envoyant (S,ida,va), tels que S : représente une requête de stockage, ida : l'identifiant de la donnée va : la valeur de cette donnée ;
2. Ce nœud redirige (S,ida,va,date,IDclient) cette requête vers CH ;
3. CH envoi (S,ida,va,date,IDclient, (lign-recevant,col-recevant)) vers les coordonnateurs des quorums et vers les autres CHs, tel que :
   Si (derniere-colonne-utilisée < c) Alors
   col-recevant <- derniere-colonne-utilisée+1 ;
   lign-recevant <- derniere-ligne-utilisée ;
   Sinon
   lign-recevant <- derniere-colonne-utilisée+1 ;
   col-recevant <-1 ;
   FinSi ;
4. En recevant ce message, les CHs le diffusent vers les coordonnateurs de quorums de leurs clusters. Chaque coordonnateur recevant ce message le diffuse vers les nœuds de son quorum ;
5. Chaque nœud recevant ce message vérifie :
   Si col-recevant est sa colonne, alors il stocke (ida,va,date) sous les coordonnées (lign-recevant,col-recevant), et renvoi une confirmation sous forme d'un ACK au coordonnateur.
   Sinon, il renvoi un échec.

6. Le coordonnateur lui aussi exécute le point (5) :
   Si col-recevant est sa colonne, il stocke (ida,va,date) sous les coordonnées (lign-recevant,col-recevant), et renvoi un ACK au CH.
   Sinon, il attend. Après avoir reçu au moins un " ACK ", il envoi " ACK " au CH.
7. CH, après avoir reçu au moins un ACK, il le renvoi au client, et met à jour la variable qui représente la dernière cellule modifiée.

4.6 Protocole de lecture

1. Client envoi une requête de lecture RL (L,ida,date, IDclint) au nœud le plus proche. Tel que : **L** : désigne la requête de lecture, **ida** : la donnée sollicité pour la lecture, **IDclient** : l'adresse du client, **date** : date de la requête.
2. À la réception de la RL :
   Scénario 2.1 : Si le nœud récepteur n'est pas un coordonnateur, alors il redirige la RL vers son coordonnateur et passe au scénario 2.2.
   Scénario 2.2 : Si le nœud récepteur est un coordonnateur, alors il exécute l'un des scénarios 2.3, 2.4, 2.5.
   Scénario 2.3 : S'il coordonne un quorum de lecture, il redirige la RL vers le CH et attend la confirmation.
   Scénario 2.4 : S'il coordonne un quorum d'écriture et appartient à un quorum de lecture, alors il redirige cette requête vers le coordonnateur du quorum de lecture, qui, à son tour, redirige la RL vers CH.
   Scénario 2.5 : S'il coordonne un quorum d'écriture et n'appartient à aucun quorum de lecture, il redirige alors la RL vers le CH.
3. Lorsque CH reçoit la RL :
   Scénario 3.1 (Dans le cas des scénarios 2.3) : S'il n y a pas de requête d'écriture en cours sur la donnée sollicité, CH envoi directement un ACK au coordonnateur. Sinon il attend que l'écriture en cours soit achevée pour envoyer un ACK au coordonnateur.
   Scénario 3.2 (Dans le cas du scénario 2.4 et 2.5) : S'il n y a pas d'écriture en cours, CH désigne un quorum de lecture le plus proche, puis envoi la RL accompagnée d'un ACK vers son coordonnateur. Sinon le CH attend l'achèvement de l'écriture en cours, puis envoie la RL accompagnée ACK au coordonnateur du quorum de lecture le plus proche qu'il désigne.
4. A la réception de l'ACK de RL (scénario 3.1) ou de RL+ACK (scénarios 3.2), le coordonnateur diffuse la RL vers les nœuds de son quorum.

5. Chaque nœud, y compris le coordonnateur, vérifie s'il a la donnée requise.
   Scénario 5.1 : Si le nœud a la données sollicitée, il l'envoi vers le coordonnateur avec son estampille.
   Scénario 5.2 : Sinon, il envoie un NACK.
6. Le coordonnateur, lorsqu'il reçoit toutes les réponses des nœuds de son quorum (données et NACKs), il compare les estampilles de ces données et envoi au CH la donnée la plus récente.
7. CH répond au client avec la version la plus récente de la donnée.

Scénarios résultants de défaillances

Dans le cas où le délai d'attente de CH est expiré (CH ne reçoit pas de réponses en présence de nœuds défaillants), il choisit un autre quorum de lecture qui lui est proche et envoi la requête de lecture, et la procédure de lecture est recommencée.
Si CH expire sa liste de coordonnateurs de quorums de lecture, et qu'il ne reçoit toujours pas de réponse positive (donnée), il envoi RL au CH le plus proche. Ainsi, le nouveau CH s'occupe de répondre au client et la procédure de lecture est recommencée.

Remarque : Le système à quorums dans le Cloud reste applicable en cas de défaillance. Nous supposons que le Cloud est toujours disponible car une solution de disponibilité a déjà été installée. Par exemple la solution proposée dans [24] qui consiste en l'exploitation de l'historique d'accès au Cloud de chaque utilisateur pendant que le Cloud est disponible. Ainsi, lorsque le Cloud ne répond pas, ces historiques sont sollicités par l'utilisateur, et ce dernier contacte un client tiers pour satisfaire sa requête.

4.7 Protocole d'écriture

1. Le client envoi une requête d'écriture RE (E,ida,va, date, IDclient,) au nœud le plus proche.
2. À la réception de la RE :
   Scénario 2.1 : Si le nœud récepteur n'est pas un coordonnateur, alors nous aurons soit le scénario 2.1.1 soit 2.1.2.
   Scénario 2.1.1 : Si le nœud récepteur appartient à un quorum de lecture, alors il redirige la RE vers son coordonnateur et passe au scénario 2.2.1.

Scénario 2.1.2 : Si le nœud récepteur appartient à un quorum d'écriture, alors il exécute l'écriture, et puis redirige la RE suivie du résultat (ACK ou NACK) vers son coordonnateur, et passe au scénario 2.2.
Scénario 2.2 : Si le nœud récepteur est un coordonnateur, alors il exécute l'un des scénarios 2.2.1, 2.2.2.
Scénario 2.2.1 : S'il coordonne un quorum de lecture, il redirige la RE vers le coordonnateur du quorum d'écriture et passe au scénario 2.2.2.
Scénario 2.2.2 : S'il coordonne un quorum d'écriture :

(a) Il redirige la RE vers le CH et vers les nœuds de son quorum, sauf le nœud recepteur (cas de Scénario 2.1.2).

(b) Chaque nœud, y compris le coordonnateur, exécute l'écriture (la mise à jour) s'il a la donnée requise, et envoi un ACK. Sinon (s'il n'a pas la donnée requise) il envoie un NACK.

(c) En parallèle, CH diffuse un message de supervision vers les coordonnateurs de quorums de son cluster (sauf celui qui lui a envoyé la RE) et vers les autres CHs.

(d) À la réception de ce message, les CHs le diffusent vers tous les coordonnateurs de quorums de leur cluster.

(e) À la réception de ce message, chaque coordonnateur le diffuse aux nœuds de son quorum, et tous les nœuds exécutent l'écriture.

Remarques :

— Un nœud, s'il trouve la donnée et avant d'exécuter l'écriture, vérifie si la date de la nouvelle requête est supérieure à celle qui est stockée. Si c'est le cas alors il exécute l'écriture.

— Le coordonnateur, s'il réussit à écrire la donnée, n'attend pas les réponses de tous les nœuds de son quorum pour envoyer un ACK au CH. Sinon, au premier ACK qu'il reçoit de ses nœuds, il le renvoi au CH, qui, à son tour, confirme l'écriture au client.

Remarque : Si un client ne reçoit pas de confirmation à sa requête(d'écriture ou de lecture) après un certain délai, il sollicite un autre noeud qui lui est proche. Ainsi, le processus de lecture ou d'écriture est recommencé.

Scénarios résultants de Défaillances

— En cas de défaillances, lorsque le délai d'attente de CH expire, il diffuse une interrogation vers tous les coordonnateurs des quorums, pour savoir s'ils ont écrit ou pas.
— Au premier ACK que le CH reçoit, il le renvoi au client.
— Si, après un délai, CH ne reçoit aucune réponse, il sollicite un autre CH qui lui est proche et lui envoi un NACK avec l'adresse du client. Ce dernier CH s'occupe de répondre au client.

4.7.1 Vérification des propriétés

— **Vivacité** :
  — Notre solution tolère plusieurs lectures simultanées sur une même donnée, une lecture ne bloque pas une autre.
  — Puisqu'une écriture ne dépend pas d'une autre écriture, alors notre solution tolère plusieurs écritures simultanées sur une même donnée.

  Donc aucun blocage ou interblocage ne peut se produire.
— **Exactitude** : Le comportement prévu au départ est toujours assuré. Si la donnée sollicitée existe, elle est toujours retournée au client, soit :
  — Par le quorum sollicité ;
  — Par les autres quorums ;
  — Au pire des cas, par les autres clusters.
— **Sureté** : Notre proposition repose sur un Cloud sûre (la perte de messages est très rare). Néanmoins, dans le cas extrême où le client sollicite un nœud défaillant, ou le coordonnateur tombe en panne avant qu'il puisse envoyer la requête au CH, nos protocoles de lecture ou d'écriture restent fonctionnels puisque le client réemet sa requête, après une attente, vers un autre nœud.

4.8 Caractéristiques de notre proposition et des travaux étudiés

À partir de l'étude des articles faites dans le chapitre 3, nous avons pu établir un tableau comparatif entre notre proposition et les articles étudiés. Il s'agit du tableau (Tab. 4.2).

Classes	Nom de la solution	Concept utilisé	Cohérence	Réplication
Performances	Harmony	Un modèle d'estimation de taux de lectures périmées	Adaptative (selon les exigences de l'application)	Asynchrone
	DcaaS	Plateforme middleware	Adaptative (selon les exigences de l'application)	Asynchrone
Coût	C3	DaaS	Adaptative (selon les transactions)	Non définie
	Aye [auteur]	L'état d'une réplique dans un Cloud privé	Forte	Asynchrone
Quorums	LibRe	Registre	Forte	Asynchrone
	Quorum circulaire	Quorums	Eventuelle forte	Asynchrone
Estampilles	PPS	Estampilles de position	Éventuelle	Asynchrone
	Diao [auteur]	Estampilles	Adaptative (selon le type de donnée)	Synchrone / asynchrone
Quorums / Estampilles	EQC2	Cluster, quorum, estampille	Eventuelle forte	Asynchrone

Table 4.2 – Caractéristiques de notre proposition et des travaux étudiés.

4.9 Comparaisons par rapport aux autres travaux

Dans cette section, nous présentons une synthèse de ce qu'apporte notre proposition par rapport aux solutions étudiées. Et ce, dans le tableau suivant (Tab. 4.3).

Classes	Nom de solution	Inconvénient	Apport de notre proposition par rapport à la solution étudiée
Performance	Harmony	Lorsqu'une application tolère un taux élevé de lectures périmées, harmony lui attribuera une cohérence éventuelle, et va alors diminuer le nombre de répliques impliquées dans la lecture. Dans ce cas nous allons rencontrer le problème de disponibilité.	Jusqu'à la convergence des répliques vers un état de cohérence, notre proposition tolère aussi des répliques périmées. La différence est qu'avec notre proposition le nombre de répliques est statique et ne va donc pas être modifié suivant les exigences de l'application, ce qui écarte le problème de disponibilité.
	DcaaS	DCaaS ne s'occupe que de la cohérence au niveau applicatif.	Notre proposition s'occupe de la cohérence au niveau du stockage, ce qui implique la cohérence au niveau applicatif.
Coût	C3	la solution C3 ne concerne que les transactions, les données peuvent être incohérentes	Notre proposition ne se consacre qu'à la cohérence des données, ce qui implique la cohérence des transactions.
	Aye [auteur]	Le système est disponible que dans le cas où la réplique est active et cohérente, dans les autres cas le système reste indisponible	Notre système reste disponible même en cas de défaillance ou d'incohérence.
Quorums	LibRe	L'écriture est réussie que lorsque tous les nœuds arrivent à écrire à leur niveau, sinon un échec est envoyé au client	Il suffit qu'un seul nœud exécute une écriture pour renvoyer un succès au client
	quorum circulaire	Lors d'une lecture un échec est directement envoyé au client si le quorum ciblé ne répond pas.	si une RL n'est pas satisfaite par le premier quorum ciblé, alors la RL sera envoyé vers d'autres quorums ensuite vers d'autres clusters jusqu'à ce que la donnée soit trouvée.

Estampilles	PPS	Lors d'une suppression d'un élément, Dans le cas de PPS, nous pouvons avoir des estampillés qui ne sont pas attribués à des d'éléments.	Dans notre proposition, chaque estampille est affectée à une donnée.
	Diao [auteur]	Difficulté du maintien de la synchronisation de la table d'horodatage dans le Cloud.	Les estampilles sont indiquées par le client, alors toutes les données auront avec la même date.

TABLE 4.3: Tableau comparatif.

4.10 Exemple illustrant notre proposition

Pour un nombre de noeuds N=9, nous aurons un nombre de colonnes c=3 et r=2, un nombre de quorums q=Max(c,r)+1=4, selon léquation 4.1. Nous avons trois quorums (q-1) de trois nœuds(c nœuds) et un quorum de quatre nœuds(c+1 nœuds). Les figures suivantes (Fig. 4.5 et 4.6) montrent l'architecture interne du cluster pour cet exemple.

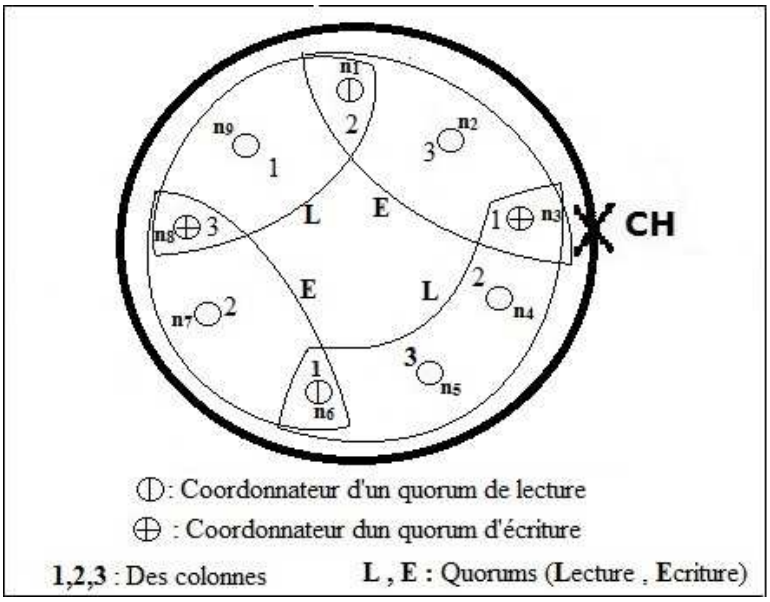

FIGURE 4.5 – Architecture du cluster pour N=9.

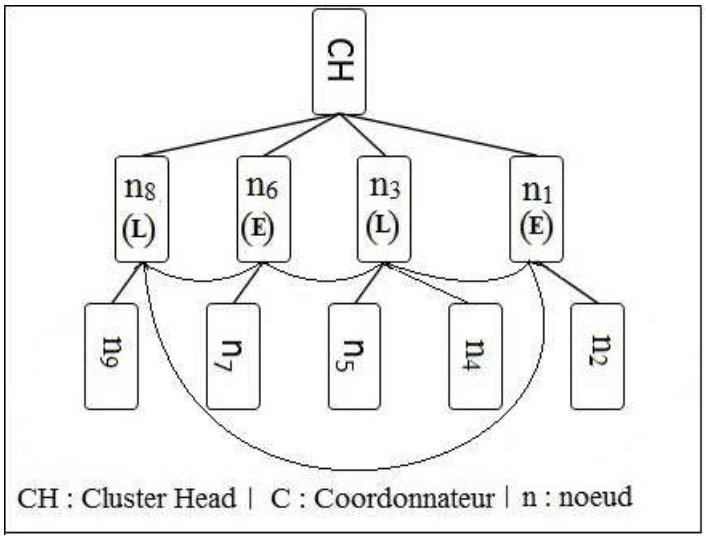

FIGURE 4.6 – Liaisons existantes dans un cluster pour N=9.

4.10.1 Scénarios d'envoi de messages pour cet exemple

Nous détaillerons les différents scenarios d'exécution de notre proposition sur la plate-forme JADE. JADE (Java Agent DEvelopement Framework) est une plate-forme multi-agent créé par le laboratoire TILAB et décrite par Belli et al. dans [1]. Elle permet le développement de systèmes multi-agents et d'applications. Elle est implémentée en JAVA et elle définit le comportement des agents.

Dans tous ce qui suit, **a** et **c** sont des noms de données, **v** la valeur de la donnée et **d** sa date(estampille). Nous supposons qu'il n y a pas eu de défaillances durant l'exécution des scénarios de lecture et d'écriture.

Les acteurs de nos protocoles y sont représentés sous forme d'agent, tels que :

Pour une lecture

- **Client :** C'est un agent représentant le client.
- **Noeud ou NoeuL :** C'est un agent représentant un nœud d'un quorum de lecture.
- **CoordL :** C'est un agent représentant le coordonnateur d'un quorum de lecture.
- **CH :** C'est un agent représentant le ClusterHead (CH).
- **CoordE :** C'est un agent représentant le coordonnateur d'un quorum d'écriture.
- **NoeuE :** C'est un agent représentant un nœud d'un quorum d'écriture.
- **AutreCH :** C'est un agent représentant les autres ClusterHeads (CH).

— **AutreCoord :** C'est un agent représentant les autres coordonnateurs(ceux du cluster sollicité par le client ou ceux qui sont dans les autres clusters).
— **AutreNoeud :** C'est un agent représentant les autres nœuds de quorums du cluster sollicité par le client ou ceux qui sont dans les autres clusters.

A noter que **Other** est un agent généré automatiquement par la plateforme JADE, il englobe les agents internes du système.

Ainsi, les représentations des protocoles sous JADE sont les suivantes.

Dans le cas d'une lecture

Dans le cas normal : Dans ce cas, la requête de lecture atteint en premier lieu un nœud d'un quorum de lecture. Dans ce cas, l'envoi de messages est illustré dans la figure (Fig. 4.7).

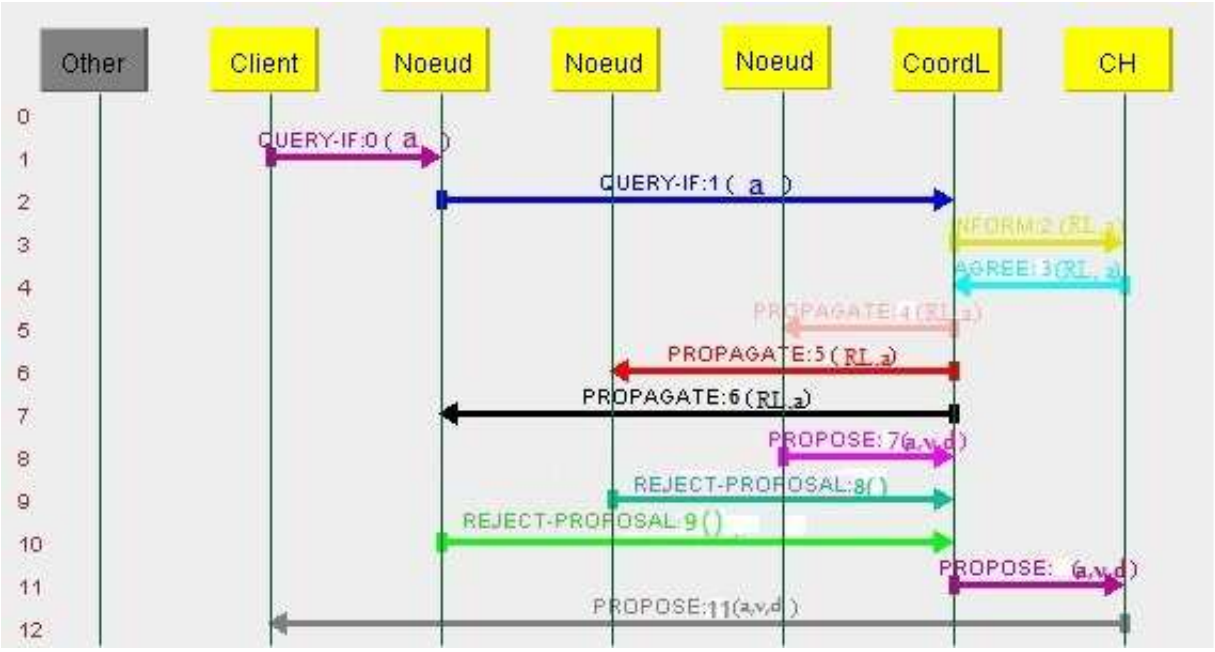

FIGURE 4.7 – Schéma d'envoi de messages lors d'une lecture dans le cas normal.

Dans le cas anormal : Dans ce cas, la requête de lecture atteint en premier lieu un nœud d'un quorum d'écriture. Pour ce cas, nous avons deux scénarios possible.

1. Soit le coordonnateur du quorum d'écriture, auquel appartient ce nœud, appartient à un quorum de lecture, et donc nous aurons un déroulement présenté dans la figure (Fig. 4.8).

2. Soit le coordonnateur du quorum d'écriture, auquel appartient ce nœud, n'appartient à aucun quorum de lecture, et donc nous aurons un déroulement présenté dans la figure (Fig. 4.9).

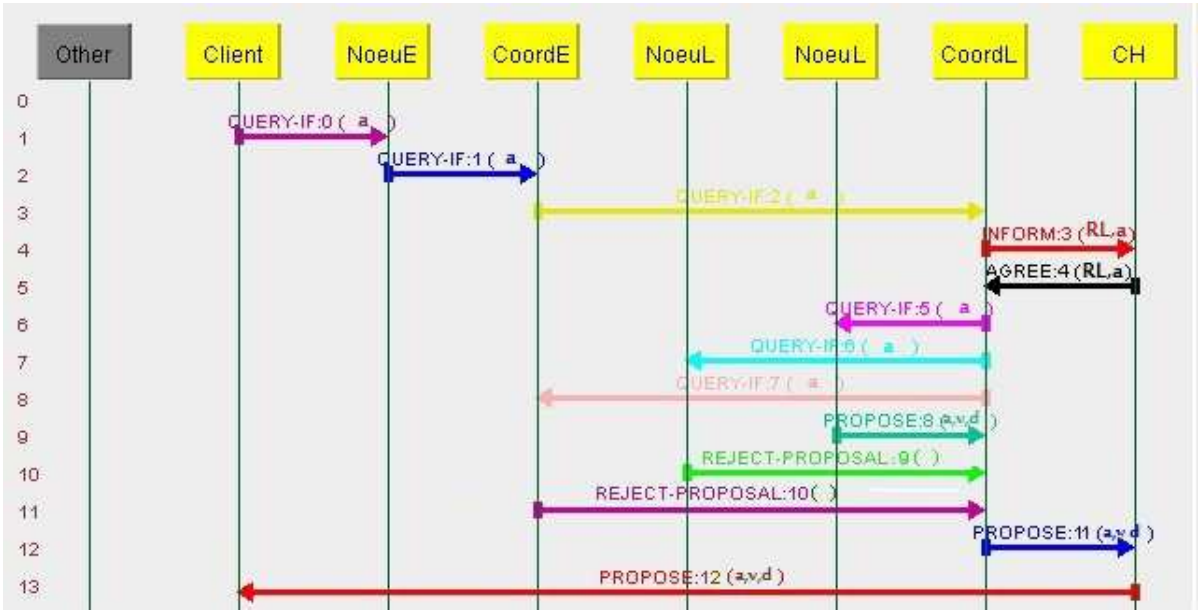

FIGURE 4.8 – Schéma d'envoi de messages lors d'une lecture dans le cas anormal (1).

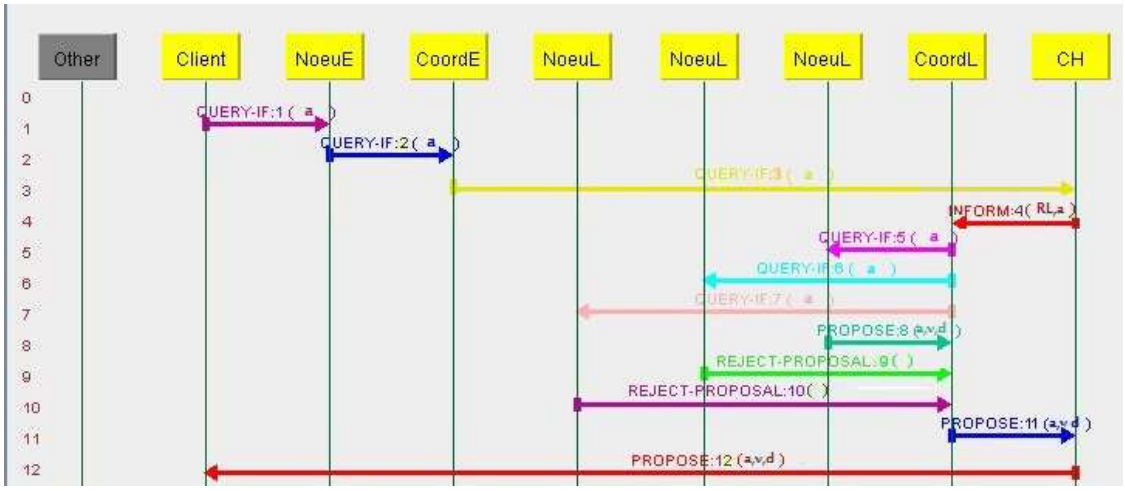

FIGURE 4.9 – Schéma d'envoi de messages lors d'une lecture dans le cas anormal (2).

Dans le cas d'une écriture

Dans le cas normal Dans ce cas, la requête d'écriture atteint en premier lieu un nœud d'un quorum d'écriture. Nous aurons ainsi l'exécution montrée dans la figure(Fig. 4.10).

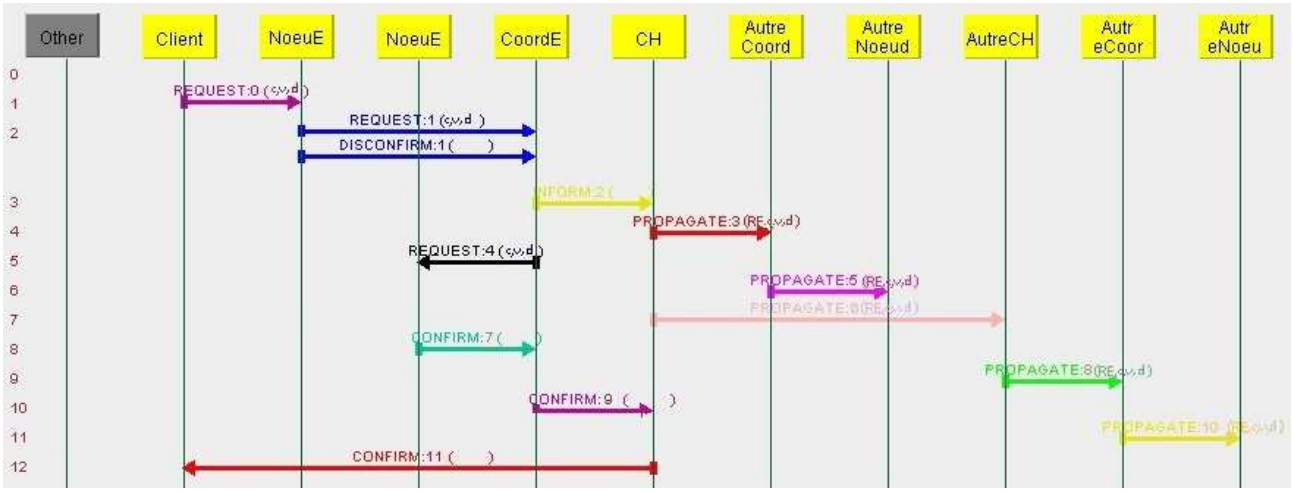

FIGURE 4.10 – Schéma d'envoi de messages lors d'une écriture dans le cas normal.

Dans le cas anormal Dans ce cas, la requête d'écriture atteint en premier lieu un nœud d'un quorum de lecture. Nous aurons ainsi l'exécution montrée dans la figure(Fig. 4.11).

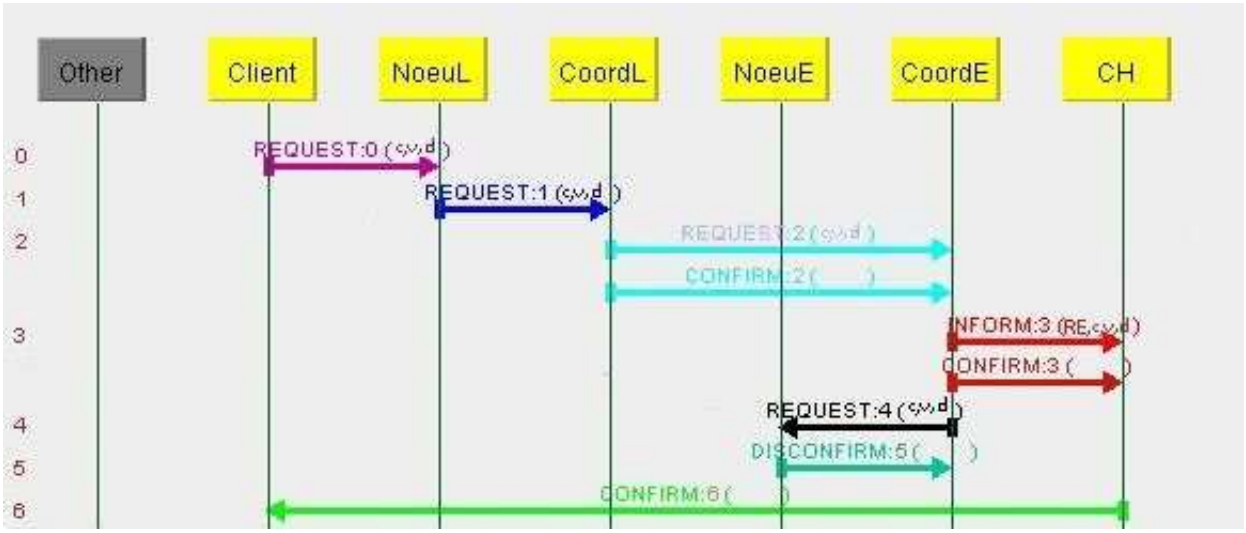

FIGURE 4.11 – Schéma d'envoi de messages lors d'une écriture dans le cas anormal.

4.11 Validation de notre proposition

4.11.1 Nombre de messages

Afin d'évaluer notre travail, nous avons déroulé plusieurs scénarios ou nous faisons varier le nombre de copies originales et de leur répliques, puis calculer le nombre de msgs générés :

- **Pour une lecture :**

  Dans le cas où c≥r :

  - **Pour (c,r)=(2,2) :** Au meilleur des cas =10=2*2+6 ; Au pire des cas = 11=2*2+7.
  - **Pour (c,r)=(3,2) :** Au meilleur des cas =10=2*2+6 ; Au pire des cas = 13=2*3+7.
  - **Pour (c,r)=(3,3) :** Au meilleur des cas =12=2*3+6 ; Au pire des cas = 13=2*3+7.
  - **Pour (c,r)=(4,3) :** Au meilleur des cas =12=2*3+6 ; Au pire des cas = 15=2*4+7.
  - **Pour (c,r)=(4,4) :** Au meilleur des cas =14=2*4+6 ; Au pire des cas = 15=2*4+7.
  - **Pour (c,r)=(5,4) :** Au meilleur des cas =14=2*4+6 ; Au pire des cas = 17=2*5+7.

  Dans le cas où c≤r :

  - **Pour (c,r)=(2,2) :** Au meilleur des cas =10=2*2+6 ; Au pire des cas = 11=2*2+7.
  - **Pour (c,r)=(2,3) :** Au meilleur des cas =10=2*2+6 ; Au pire des cas = 11=2*2+7.
  - **Pour (c,r)=(3,3) :** Au meilleur des cas =12=2*3+6 ; Au pire des cas = 13=2*3+7.
  - **Pour (c,r)=(3,4) :** Au meilleur des cas =12=2*3+6 ; Au pire des cas = 13=2*3+7.
  - **Pour (c,r)=(4,4) :** Au meilleur des cas =14=2*4+6 ; Au pire des cas = 15=2*4+7.
  - **Pour (c,r)=(4,5) :** Au meilleur des cas =14=2*4+6 ; Au pire des cas = 15=2*4+7.

- **Pour une écriture, sans diffusions :**

  Dans le cas où c=r :

- **Pour (c,r)=(2,2) :** Au meilleur des cas =8=2(2+2) ; Au pire des cas = 9=2(2+2)+1.
- **Pour (c,r)=(3,3) :** Au meilleur des cas =10=2(3+2) ; Au pire des cas = 11=2(3+2)+1.
- **Pour (c,r)=(4,4) :** Au meilleur des cas =12=2(4+2) ; Au pire des cas = 13=2(4+2)+1.

Dans le cas où c≠r :

- **Pour (c,r)=(2,3) :** Au meilleur des cas =8=2(2+2) ; Au pire des cas = 9=2*2+5.
- **Pour (c,r)=(3,4) :** Au meilleur des cas =10=2(3+2) ; Au pire des cas = 11=2*3+5.
- **Pour (c,r)=(4,5) :** Au meilleur des cas =12=2(4+2) ; Au pire des cas = 13=2*4+5.
- **Pour (c,r)=(3,2) :** Au meilleur des cas =8=2(2+2) ; Au pire des cas = 9=2*2+5.
- **Pour (c,r)=(4,3) :** Au meilleur des cas =10=2(3+2) ; Au pire des cas = 11=2*3+5.
- **Pour (c,r)=(5,4) :** Au meilleur des cas =12=2(4+2) ; Au pire des cas = 13=2*4+5.

— **Pour une écriture, le nombre de messages en plus qui est résultant de la diffusion :**

Dans le cas où c=r :

- **Pour (c,r)=(2,2) :** À rajouter 6 messages(soit 2*(2+1)=6=Nombre de nœuds **N**).
- **Pour (c,r)=(3,3) :** À rajouter 12 messages(soit 3*(3+1)=12=Nombre de nœuds **N**).
- **Pour (c,r)=(4,4) :** À rajouter 20 messages(soit 4*(4+1)=20=Nombre de nœuds **N**).

Dans le cas où c≠r :

- **Pour (c,r)=(2,3) :** À rajouter 9 messages(soit 2*(3+1)+1=N+1).
- **Pour (c,r)=(3,4) :** À rajouter 16 messages(soit 3*(4+1)+1=N+1).
- **Pour (c,r)=(4,5) :** À rajouter 25 messages(soit 4*(5+1)+1=N+1)
- **Pour (c,r)=(3,2) :** À rajouter 10 messages(soit 3*(2+1)+1=N+1)
- **Pour (c,r)=(4,3) :** À rajouter 17 messages(soit 4*(3+1)+1=N+1)
- **Pour (c,r)=(5,4) :** À rajouter 26 messages(soit 5*(4+1)+1=N+1)

4.11.2 Généralisation

À partir des résultats de la section précédente (Section 4.11.1), en procédant par réccurence, nous trouvons (soient **m** le nombre de messages dans le meilleur des cas, et **p** dans le pire des cas) :

— **Le nombre de messages pour une lecture :** Nous avons deux cas de figure : soit c≤r, et donc nous obtenons l'équation 4.2, soit c≥r, et donc nous obtenons l'équation 4.3.

$$c \leq r \Rightarrow (m = 2 * c + 6) \wedge (p = 2 * c + 7) \tag{4.2}$$

$$c \geq r \Rightarrow (m = 2 * r + 6) \wedge (p = 2 * c + 7) \tag{4.3}$$

— **Le nombre de messages pour une écriture :** Nous avons obtenu un nombre de messages sans diffusion (équation 4.4), et avec diffusion (équation 4.5 lorsque c=r et équation 4.6 lorsque c≠r).

$$(m = 2 * Min(c, r) + 2) \wedge (p = 2 * Min(c, r) + 5) \tag{4.4}$$

$$c = r \Rightarrow (m = 2 * Min(c, r) + c(r + 1) + 2) \wedge (p = 2 * Min(c, r) + c(r + 1) + 5) \tag{4.5}$$

$$c \neq r \Rightarrow (m = 2 * Min(c, r) + c(r + 1) + 3) \wedge (p = 2 * Min(c, r) + c(r + 1) + 6) \tag{4.6}$$

Preuve Dans tous les cas, le nombre de message, s'il augmente, augmente de 2 à chaque fois qu'un nœud est ajouté, puisqu'il ya deux messages en plus (celui qu'il reÃ§oit et celui qu'il envoi).

1. **Nous allons démontrer le nombre de message pour une lecture :**
   — **Par réccurence, démontrons que c≤r⇒(m=2*c+6)∧(p=2*c+7).**
   Pour (c,r)=(2,3), nous avons m=10=2*2+6 et p=11=2*2+7 : L'équation est vérifiée pour (c,r)=(2,3).
   Pour un c et un r tel que c≤r, nous suposons que m=2*c+6 et p=2*c+7 et nous démontrons par récurrence que p'=2*c'+7 et m'=2*c'+6, tel que c'=c+1.
   Nous savons que le nombre de messages augmente de 2, donc p_{n+1}=p_n+2 et m_{n+1}=m_n+2.
   Donc **p'**=p+2=2*c+7+2=2*(c+1)+7**=2*c'+7**, et **m'**=m+2=2*c+6+2=2*(c+1)+6**=2*c'+6**.
   D'où, l'équation c≤r⇒(m=2*c+6)∧(p=2*c+7) est vérifiée (pour une lecture).
   — **Par réccurence, démontrons c≥r⇒(m=2*r+6)∧(p=2*c+7) pour une lecture.**
   Pour (c,r)=(3,2), nous avons m =10=2*2+6 et p=13=2*3+7 : l'équation est

vérifiée pour (c,r)=(3,2).

Pour un c et un r tel que c≥r, nous suposons que m=2*r+6 et p=2*c+7 et nous démontrons par récurrence que p'=2*c'+7 et m'=2*r'+6, tel que c'=c+1 et r'=r+1.

Nous savons que le nombre de messages augmente de 2, donc p_{n+1}=p_n+2 et m_{n+1}=m_n+2.

Donc **p'**=p+2=2*c+7+2=2*(c+1)+7**=2*c'+7**, et **m'**=m+2=2*r+6+2=2*(r+1)+6**=2*r'+6**.

D'où, l'équation c≥r⇒(m=2*r+6)∧(p=2*c+7) est vérifiée (pour une lecture).

2. **Pour une écriture :**

Démontrons par récurrence que (m=2*(Min(c,r)+2)∧(p=2*Min(c,r)+5)) pour une écriture, quand (c≠r) et (c=r).

Pour (c,r)=(2,2) : m =8=2(2+2) et p=9=2*2+5. Et pour (c,r)=(2,3) : m=8=2(2+2) et p = 9=2*2+5. Et pour (c,r)=(3,2) : m=8=2(2+2) ; p = 9=2*2+5

D'où, l'équation est vérifiée pour (c,r)=(2,2), (c,r)=(2,3) et pour (c,r)=(3,2).

Pour un c et un r tels que soit c=r, soit —c-r—=1, nous suposons que (m = 2 âˆ— (Min(c, r) + 2) âˆ§ (p = 2 âˆ— Min(c, r) + 5)) et nous démontrons par récurrence que m' = 2 âˆ— (Min(c', r') + 2) âˆ§ (p' = 2 âˆ— Min(c', r') + 5), tel que c'=c+1 et r'=r+1 .

Nous savons que le nombre de messages augmente de 2, donc mn+1=mn+2 et pn+1 =pn+2.

Et donc m'=m+2=2*(Min(c,r)+2)+2=2*(Min(c,r)+1+2)=2*(Min((c+1),r+1)+2)=2*(Min(c',r')+2

et p'=p+2=2 âˆ— Min(c, r) + 5+2=2âˆ— (Min(c, r) +1)+ 5=2 âˆ— Min((c+1), r+1) + 5=2âˆ— Min(c', r') + 5.

D'où, (m = 2 âˆ— (Min(c, r) + 2) âˆ§ (p = 2 âˆ— Min(c, r) + 5)) est vérifiée pour (c=r) et (c≠r).

4.12 Évaluation des Performances

4.12.1 Temps moyen de réponse

Nous supposons que le temps d'envoi sur une liaison =1 unité de temps, le temps de traitement de requêtes ou de données=1u, le temps de traitement d'ACK ou de NACK=0.5u.

Nous avons obtenu les résultats suivants :

— **Pour une lecture :**

- **Pour (c,r)=(2,2) :** Au meilleur des cas =11.5u ; Au pire des cas =12.5u.
- **Pour (c,r)=(3,2) :** Au meilleur des cas =11.5u ; Au pire des cas =12.5u.
- **Pour (c,r)=(2,3) :** Au meilleur des cas =11.5u ; Au pire des cas =12.5u.
- **Pour (c,r)=(3,3) :** Au meilleur des cas =11.5u ; Au pire des cas =12.5u.
- **Pour (c,r)=(4,3) :** Au meilleur des cas =11.5u ; Au pire des cas =12.5u.
- **Pour (c,r)=(3,4) :** Au meilleur des cas =11.5u ; Au pire des cas =12.5u.
- **Pour (c,r)=(4,4) :** Au meilleur des cas =11.5u ; Au pire des cas =12.5u.
- **Pour (c,r)=(5,4) :** Au meilleur des cas =11.5u ; Au pire des cas =12.5u.
- **Pour (c,r)=(4,5) :** Au meilleur des cas =11.5u ; Au pire des cas =12.5u.

— **Pour une écriture, sans diffusions :**

- **Pour (c,r)=(2,2) :** Au meilleur des cas =6.5u ; Au pire des cas =11u.
- **Pour (c,r)=(3,3) :** Au meilleur des cas =6.5u ; Au pire des cas =11u.
- **Pour (c,r)=(4,4) :** Au meilleur des cas =6.5u ; Au pire des cas =11u.
- **Pour (c,r)=(2,3) :** Au meilleur des cas =6.5u ; Au pire des cas =11u.
- **Pour (c,r)=(3,4) :** Au meilleur des cas =6.5u ; Au pire des cas =11u.
- **Pour (c,r)=(4,5) :** Au meilleur des cas =6.5u ; Au pire des cas =11u.
- **Pour (c,r)=(3,2) :** Au meilleur des cas =6.5u ; Au pire des cas =11u.
- **Pour (c,r)=(4,3) :** Au meilleur des cas =6.5u ; Au pire des cas =11u.
- **Pour (c,r)=(5,4) :** Au meilleur des cas =6.5u ; Au pire des cas =11u.

Nous remarquons que les temps de réponse pour les deux opérations (lecture et écriture) demeurent stable. Ceci est dû au temps infinitésimal de traitement d'une requête ou d'une donnée par le datacenter qui représente le coordonnateur du quorum, et aussi grâce au parallelisme existant du traitement et d'envoi.

Même en cas de diffusion pour une opération d'écriture, nous obtenons dans le meilleur des cas, Tm=9u, et donc notre proposition tolère une lecture périmée durant 2.5u (2.5=9-6.5).

4.12.2 Représentation graphique

Dans cette section, nous allons présenter les graphes des nombres de messages et du temps de réponse avec MATLAB. MATLAB est dérivé de l'anglais Matrix Laboratory, c'est une application scientifique orientée calcul matriciel et vectoriel avec une puissante librairie de virtualisation. MATLAB aide à résoudre des problèmes de calculs très complexes de façon simple et rapide comparée aux langages de programmation traditionnels de type C ou Fortran. Il permet d'afficher des courbes qui rendent facile à visualiser et acquérir des connaissances à partir de données, met en œuvre des algorithmes et permet la création des IHMs [8].

Et lors d'une écriture et en l'absence de diffusion c a d que le msg n'est pas diffusé aux

autres quorums du cluster : nous obtenons ces formules dans les cas où c≥r et c<r et dans le meilleur et pire des cas

Les résultats obtenus dans la section 4.11.1 sont représentés comme suit :

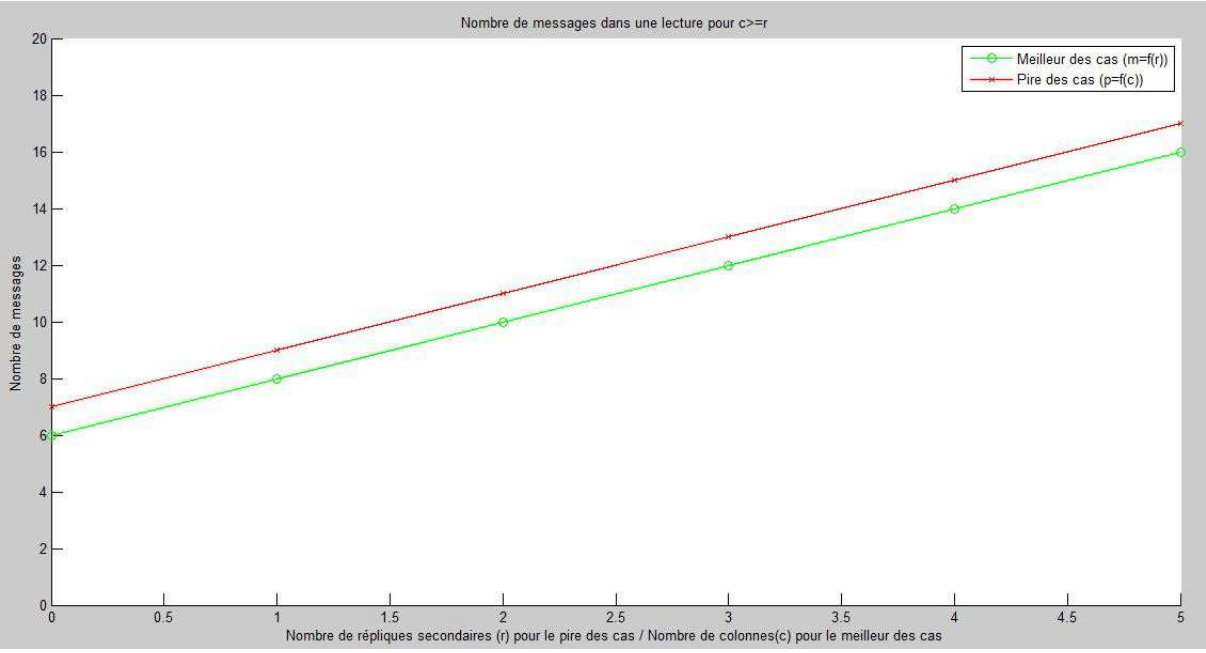

FIGURE 4.12 – Nombre de messages pour une opération de lecture.

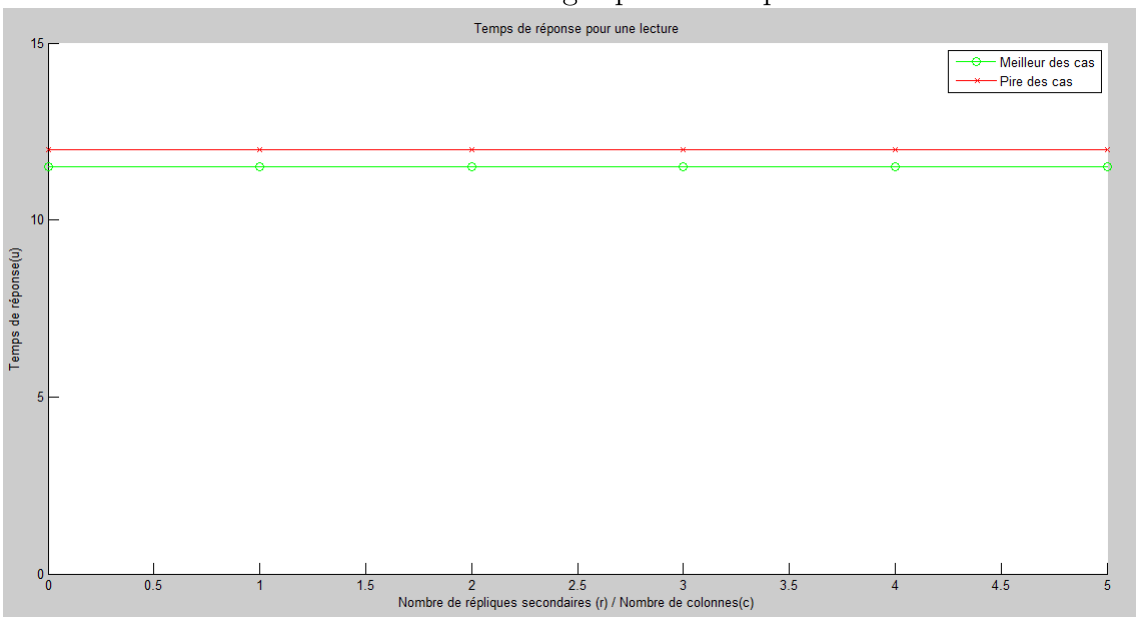

FIGURE 4.13 – Temps de réponse d'une lecture.

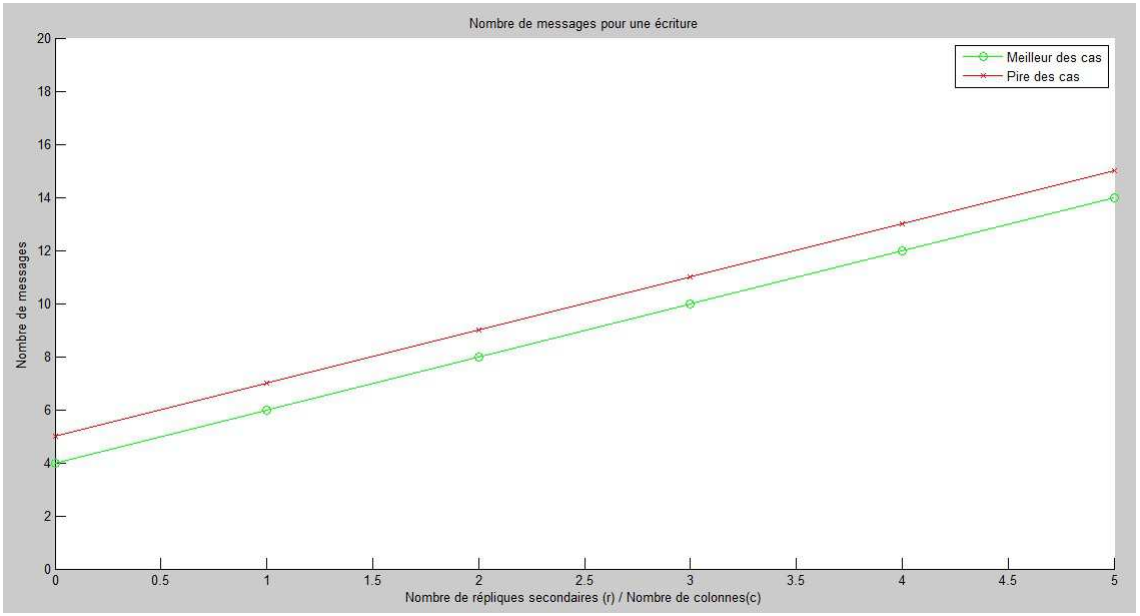

FIGURE 4.14 – Nombre de messages pour une opération d'écriture.

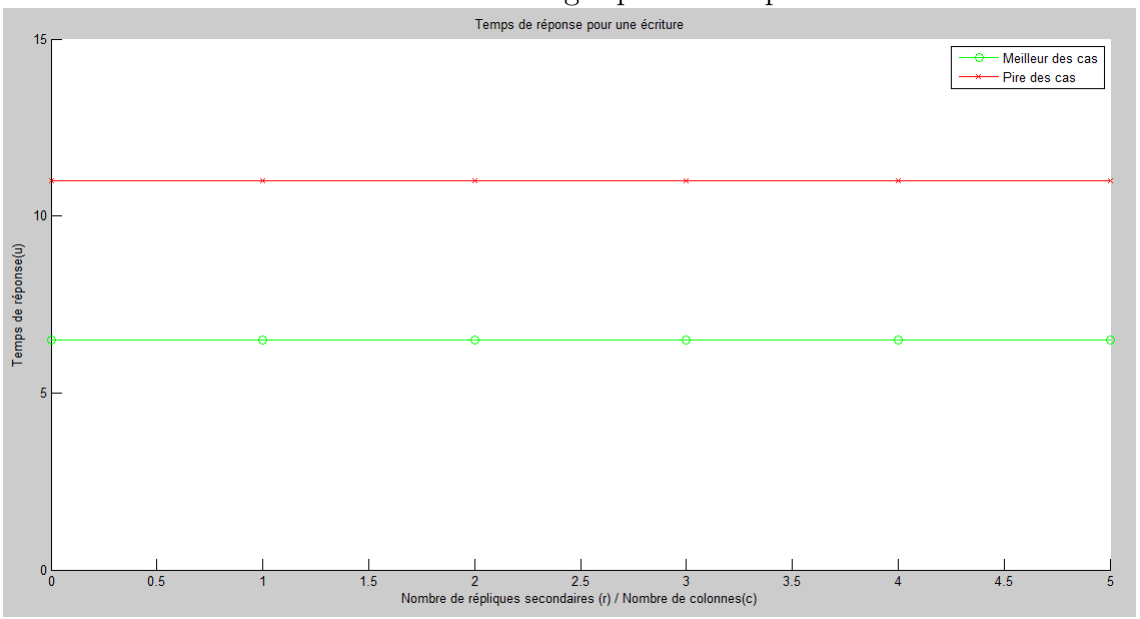

FIGURE 4.15 – Temps de réponse d'une écriture sans diffusions.

Nous remarquons que le graphe correspondant au nombre de message sous Matlab dans le cas d'une lecture et écriture se traduit par une droite linéaire qui croit avec le nombre de nœuds. De plus, nous remarquons que même avec le nombre de nœuds qui croit, le temps de réponse reste constant dans le meilleur et pire des cas et cela est dû au parallélisme. En effet, lorsque le CH et un coordonnateur d'un quorum diffuse un message, peu importe le nombre de nœuds, l'envoi se fait en parallèle.

Conclusion

Nous avons proposé une solution indépendante des autres travaux fait sur notre thème. En effet, nous avons défini des protocoles d'écriture et de lecture et passé en revue tous les scénarios possibles á notre proposition, validé et évalué les performances de notre solution.

Conclusion et perspectives

La diversité, la vitesse et le volume des données d'aujourd'hui n'ont jamais été aussi élevés, ce phénomène est appelé Big Data. La gestion de Big Data devient difficile avec les outils classiques. Le Cloud représente l'infrastructure la mieux adaptée, et cela grâce aux services qu'il offre. Afin de faire face aux défis de Big Data, les données sont répliquées dans des sites dispersés. Cependant, la réplication introduit le problème important de la cohérence des données.

De nombreux travaux ont été proposé pour pallier le problème cité ci-dessus, nous les avons départagés en quatre classes : les estampilles des données, les quorums, le coût et les performances des applications. La plupart de ces travaux assurent une cohérence adaptive suivant les exigences de l'application.

Tout au long de ce semestre, nous avons tenté d'apporter notre contribution pour pallier le problème de cohérence. Nous avons opté pour un Cloud clustérisé contenant des quorums de lecture et d'écriture. Chaque quorum est doté d'un coordonnateur et chaque cluster d'un clusterhead. Lors d'une requête d'écriture (mise à jour), le client va solliciter un seul quorum (le plus proche) qui s'occupera de lui répondre via le clusterhead, et cela, au lieu d'attendre la propagation de la requête vers tous les autres quorums ; ce qui permettra d'une part de donner une réponse rapide à la requête du client et de l'autre permettre au système de rester disponible pour toute lecture ultérieure. En effet dès que le premier quorum sollicité répond au clusterhead, l'écriture est considérée comme réussie (elle n'est plus en cours), et le système pourra alors traiter les requêtes suivantes. Lors d'une requête de lecture, les données vont être comparées par les coordonnateurs de chaque quorum puis par le clusterhead grâce à leur estampille, offrant ainsi au client la version la plus récente de la donnée.

Nous avons comparé notre solution avec les travaux étudiés, montrant ainsi les avantages de notre proposition par rapport à la leur.

Avec la plateforme JADE, nous avons implémenté différents scénarios se présentant à

notre solution. Nous l'avons ensuite validé, puis évalué ses performances grâce à la plateforme MATLAB.

Pour notre proposition, les quorums des clusters de notre Cloud sont statiques. Alors comme première perspective, nous proposons que les quorums soient dynamiques, autrement dit, permettre à des nœuds de rejoindre les quorums des clusters du Cloud. Nous souhaitons aussi limiter le nombre de réémissions de la part du client. Une autre perspective consiste à implémenter notre solution sur un Cloud réel afin de mieux la valider et évaluer ses performances. Ou bien, simuler notre solution sur un réseau en utilisant JADE.

Bibliographie

[1] http://aws.amazon.com/. Consulté le 28 Mai 2016.

[2] http://blogs.msdn.com/b/csliu/archive/2009/06/26/implementing_consistency_protocols_for_data_replication_and_cache_coherence.aspx. Consulté le 16 Décembre 2015.

[3] https://www.techopedia.com/definition/1236/replication. Consulté le 03 octobre 2015.

[4] http://www.computerweekly.com/feature/Write_through_write_around_write_back_Cache_explained. Consulté le 25/04/2016.

[5] http://www.figer.com/Publications/nuage.htm#.VnP6JvnhDIU. Consulté le 19 Décembre 2015.

[6] http://www.fipa.org/specs/fipa00001/. consulté le 20/06/2016.

[7] http://www.ibm.com/cloud-computing/us/en/what-is-cloud-computing.html. Consulté le 21 septembre 2015.

[8] http://www.mathworks.com/products/matlab/. Consulté le 26 Juin 2016.

[9] http://www.sciencefocus.com/qa/how-many-terabytes-data-are-internet. Consulté le 18/06/2016.

[10] http://www.spoonylife.org/algorithms-and-computation/methodes-de-clustering. Consulté le 2 Juillet 2016.

[11] http://www.techniques-ingenieur.fr/actualite/high-tech-thematique_193/nuages-sur-le-cloud-article_61412/. Consulté le 17 octobre 2015.

[12] http://www.windowsazure.com/. Consulté le 26 Mai 2016.

[13] Y.N. Aye. Data consistency on private cloud storage system. In *International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)*, pages 101 – 105, 2012.

[14] G. Chesnote. *Cloud Computing, Big Data, Parallélisme, Hadoop : Stockage de données du futur*. Vuibert, Paris, Novembre 2012.

[15] H. Chihoub, S. Ibrahim, and G. Antoniu et M.S. Pérez. Harmony : Towards automated self-adaptive consistency in cloud storage. In *the proceedings of the 2012 IEEE International Conference on Cluster Computing (Cluster'12)*, pages 293 – 301, Beijing, Septembre 2012. IEEE.

[16] H.E Chihoub. *Managing Consistency for Big Data Applications on Clouds : Tradeos and Self Adaptiveness. Distributed, Parallel, and Cluster Computing*. PhD thesis, Université européenne de Bretagne, Décembre 2013.

[17] Z. Diao. Consistency models for cloud-based online games : the storage system's perspective. In *25th GI-Workshop on Foundations of Databases*, pages 6–11, Ilmenau (Allemagne), 2013.

[18] M. Durut. *Algorithmes de classification répartis sur le cloud*. PhD thesis, TELECOM ParisTech, institut des sciences et de technologies, Septembre 2012.

[19] I. Elgedawy. Dcaas : Data consistency as a service for managing data uncertainty on the clouds. *(IJACSA) International Journal of Advanced Computer Science and Applications*, 4(5) :59–68, 2013.

[20] V. Kumar et A. Agarwal. An efficient read dominant data replication protocol under serial isolation using quorum consensus approach. 2014.

[21] J.P. Briffaut et F. Stefan. *Cloud computing : Evolution technologique, révolution des usages*. Management et informatique. Lavoisier, 2013.

[22] I. Fetai et H.Schuldt. Cost-based data consistency in a data-as-a-service cloud environnement. In *IEEE Fifth International Conference on Cloud Computing*, volume 38, pages 526–533. IEEE, 2012.

[23] B. Panchal et R.K. Kapoor. Performance enhancement of cloud computing using clustering. *IJCSNS International Journal of Computer Science and Network Security*, 14(6) :37–40, Juin 2014.

[24] L. ALOUACHE et S. AIT MESBAH. Solution de disponibilité dans les environnements cloud computing. Mémoire de master, Université A/Mira, Béjaia, 2015.

[25] A. Coutty et T. Vonfelt. *L'évolution maitrisée vers le IaaS/PaaS*. EuroCloud France, Paris, Novembre 2011.

[26] J. Gray. A transaction model. Rapport technique, IBM Research Laboratory, 1980.

[27] K. Hwang, G.C. Fox, and J.J. Dongarra. *Distributed and Cloud Computing : From Parallel Processing to the Internet of Things*. Elsevier, Massachusetts - USA, 2012.

[28] S.P. Kumar, S. Lefebvre, and R. Chiky et E. Gressier-Soudan. Calibre : A better consistency-latency tradeoff for quorum based replication systems. In *Database and Expert Systems Applications : 26th International Conference, DEXA Part II*, pages 491 – 503, Suisse, Septembre 2015. Springer.

[29] L. Lamport. How to make a multiprocessor computer that correctly executes. In *IEEE Transactions on Computers*, volume 28, page 690–691. IEEE, 1979.

[30] D. Nkongolo. Etude d'une réplication symétrique asynchrone dans une base de données répartie. application à l'enrôlement des électeurs, 2011. Mémoire de licence, université de Kinshasa.

[31] C. Pierkot. *Gestion de la Mise à Jour de Données Géographiques Répliquées.* PhD thesis, Université Toulouse III - Paul Sabatier, Juillet 2008.

[32] N. Preguica, J. M. Marques, and M. Shapiro et M. Letia. A commutative replicated data type for cooperative editing. In *ICDCS '09*, pages 395 – 403, Washington DC, 2009.

[33] W. Qinyi and P. Calton et J. E. Ferreira. A partial persistent data structure to support consistency in real-time collaborative editing. In *IEEE 26th International Conference on Data Engineering (ICDE 2010)*, page 776 – 779, Mars 2010.

[34] M. Raynal. *Gestion des données réparties : problèmes et protocoles.* Eyrolles, Paris, 1992.

[35] L. Rong. Multimedia resource replication strategy for a pervasive peer-to-peer environment. *Journal Of Computers*, 3(4) :9 – 15, Avril 2008.

[36] S. Sakr, A. Liu, and D.M. Batista et M. Alomari. A survey of large scale data management approaches in cloud environments. In *IEEE Communications Surveys and Tutorials*, volume 13, pages 311–336, 2011.

[37] M. Shapiro, N. Preguica, and C. Baquero et M. Zawirski. *The Distributed Computing Column*, volume 1, chapter Convergent and Commutative Replicated Data Types, pages 66 – 88. EATCS (European Association for Theoratical Computer Science), Juin 2011.

[38] M. Shapiro, N. Preguiça, and C. Baquero et M. Zawirski. Conflict-free replicated data types. Rapport de recherche 7687, Institut National de Recherche en Informatique et en Automatique, Juillet 2011.

[39] M. Shapiro, N. Preguiça, and C.B. Moreno et M. Zawirsky. Technical report.

RÉSUMÉ

Les données générées chaque jour dans le monde entier, que ça soit dans les réseaux sociaux, dans les entreprises ou autres, ne cessent d'augmenter. Ce phénomène est appelé Big Data. Les Big Data deviennent difficiles à gérer, et Le Cloud Computing, grâce aux services qu'il offre, reste l'infrastructure la mieux adaptée à ce phénomène. Dans ce contexte, la réplication est un moyen essentiel dans le Cloud afin de surmonter les défis de Big Data. Cependant, la réplication introduit le problème important de la cohérence des données à travers différentes copies. La gestion de la cohérence est une question critique pour les systèmes Big Data.

Dans ce mémoire, nous avons parlé de Big Data, vu des généralités sur le Cloud, introduit les modèles existants de la réplication et la cohérence dans la littérature, puis, nous avons étudié différentes approches pour pallier le problème de cohérence. Nous avons ensuite présenté notre propre solution, qui s'appuie sur le principe des quorums et qui vise à assurer une cohérence éventuelle forte. De plus, une représentation de nos protcoles sous la plateforme JADE est présentée. Nous avons finalement validé notre proposition puis vérifier les performances sous la plateforme MATLAB.

Mots clés : Big Data, Cloud Computing, Cluster, Datacenter, Quorum, Réplication, Cohérence.

ABSTRACT

The data generated every day around the world, whether in social networks, businesses or other, are increasing. This phenomenon is called Big Data. Big Data becomes difficult to manage, and Cloud Computing, through the services it offers, remains the infrastructure the best suited to this. In this context, replication is essential in order to deal with Big Data challenges. However, replication introduces the major issue of data consistency across different copies. Consistency management is a critical matter for Big Data systems.

In this master report, we talked about Big Data, seen generalities on Cloud, introduced the existing models of replication and consistency in the literature, then, we studied different approaches to overcome the problem of consistency. We then presented our own solution, based on the principle of quorums and which aims to ensure an eventual strong consistency. Also, we gave a graphical diagrams of our protocols, done with JADE platform. We finally approved our proposal and verify performances under MATLAB platform.

Key words : Big Data, Cloud Computing, Cluster, Datacenter, Quorum, Replication, Consistency.