RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITÉ LARBI TEBESSI, TEBESSA
Département de Mathématiques et Informatique
Laboratoire de Mathématiques, d'Informatique et Systèmes (LAMIS)

Thèse

Présentée pour l'obtention du diplôme de

DOCTORAT LMD
en
Informatique

Réplication de données dans les systèmes Cloud : Application pour les projets territoriaux

Spécialité : **Systèmes d'information coopératifs**

Présentée par

TABET Khaoula

Jury

Dr. DERDOUR Makhlouf	Université Larbi Tebessi, Tebessa	Président
Pr. GHALEM Belalem	Université d'Oran 1	Examinateur
Dr. HAMROUNI Tarek	Université El Manar, Tunis	Examinateur
Dr. AMROUNE Mohamed	Université Larbi Tebessi, Tébessa	Examinateur
Pr. LAOUAR Mohamed Ridda	Université Larbi Tebessi, Tébessa	Directeur de thèse
Dr. MOKADEM Riad	Université Toulouse III, France	Encadrant
Pr. BENDJENNA Hakim	Université Larbi Tebessi, Tébessa	Invité

THESE

Dédicace

Je dédie ce modeste travail,

A Mon très Cher Père

" *Je me rappellerai toujours de tous les moments où tu m'as poussé à travailler et à réussir., Cher père j'avoue que si je suis aujourd'hui à ce niveau et à ce degré de réussite c'est grâce à tes efforts, à ton sens du sacrifice et à tes conseils. J'avoue vraiment et je reconnais que tu étais et demeures pour moi la lumière qui guide mon chemin et qui m'emmène tout droit vers la réussite . Merci et j'espère que tu trouveras dans ce travail l 'expression de ma gratitude et mon respect****."***

"A celle qui était toujours présente pour moi, à tout moment et en toutes circonstances ; celle qui m'a toujours encouragée et soutenu moralement, à *ma très* ***chère mère" ;***

"A mes très chers sœurs et frère avec qui j'apprends toujours de la vie, des acquis que je n'aurais jamais su avoir sans vous, ***Amina****,* ***Nadia****,* ***Soumaya, Haroun";***

"A mes chère amies ***Roumaissa, Selma, Radia, Meriem, Imen, Aycha, Nadira.... ";***

A la mémoire de ma sœur Saida:

"Ma petite sœur, comme le temps passe, la douleur en moi reste vivace car rien ne peut combler le vide ainsi que l'infinie tristesse que tu laisses dans notre maison. Tu es partie, comme un jour d'été laissant place à la pluie dans mon cœur affligé, et rejoignant un monde de paix éternelle. Repose en paix, je ne t'oublierai jamais "

KHAOULA

Remerciements

Je tiens premièrement à me prosterner en remerciant Allah le tout-puissant de m'avoir donné le courage et la patience afin de terminer ce travail.

À l'issue de la rédaction de cette recherche, je suis convaincue que la thèse est loin d'être un travail solitaire. En effet, je n'aurais jamais pu réaliser ce travail doctoral sans le soutien de quelques personnes.

En premier lieu, je tiens à remercier mon directeur de thèse, monsieur ***LAOUAR Mohamed Ridha*** *Professeur à l'Université de Tébessa et directeur du laboratoire LAMIS-Tébessa, pour la confiance qu'il m'a accordée en acceptant d'encadrer ce travail doctoral, pour sa grande disponibilité et ses encouragements tout au long de la préparation de cette thèse. Je veux, à cet effet, lui adresser tous mes sincères remerciements pour la confiance accordée ainsi que la grande liberté d'idées et de travail qu'il m'a donné.*

Je souhaiterais exprimer ma gratitude à mon encadrant, Monsieur ***MOKADEM Riad*** *maître de conférences à l' Université Paul Sabatier (Toulouse III), et qui mène parallèlement des recherches à l'Institut de recherche en informatique de Toulouse (IRIT), au sein de l'équipe Pyramide, pour ses multiples conseils et pour toutes les heures qu'il a consacré à diriger cette thèse. J'aimerais également lui dire à quel point j'ai apprécié sa grande disponibilité et son respect sans failles des délais serrés de relecture des documents que je lui ai adressés. Enfin, j'ai été extrêmement sensible à ses qualités humaines d'écoute et de compréhension tout au long de ce travail doctoral.*

Je remercie le plus sincèrement possible les membres de jury : ***Dr. DERDOUR Makhlouf, Pr. GHALEM Belalem, Dr. HAMROUNI Tarek , Dr. AMROUNE Mohamed, et Pr. BENDJENNA Hakim*** *pour leur attention et l'intérêt porté à cette thèse et qui ont eu l'obligeance d'accepter de juger ce travail. Comme je présente mes sincères remerciements à tous les membres de LAMIS qu'ils soient professeurs, docteurs et/o doctorants qui ont contribué à élaborer ce laboratoire de recherche et qui ont su rendre mon travail agréable à travers leurs conseils, leur simple présence ainsi que les différentes journées doctorales et les conférences élaborées par eux.*

Résumé

Ces dernières années, les données sont devenues de plus en plus complexes et volumineuses, en raison de l'évolution rapide des technologies et des nouvelles applications à grande échelle. La gestion de ces données représente un défi important. Les fournisseurs de cloud visent à maximiser leurs profits tout en satisfaisant les exigences des clients, par exemple, les performances.

Les systèmes de gestion de bases de données relationnelles font face à de nombreux obstacles pour atteindre des objectifs tels que les performances. Par conséquent, l'utilisation de bases de données NoSQL devient nécessaire pour traiter des charges de travail hétérogènes et des données volumineuses.

Dans ce contexte, la réplication de données est une technique bien connue qui vise à: (i) augmenter la disponibilité de données, (ii) réduire les coûts d'accès aux données ce qui permet d'améliorer les performances et (iii) assurer une meilleure tolérance aux pannes. Néanmoins, répliquer les données sur tous les nœuds est une solution non réaliste vu qu'elle génère une consommation importante de la bande passante en plus de l'espace limité de stockage. Définir des stratégies de réplication constitue la solution à apporter à ces problématiques.

Dans ce travail, nous proposons une nouvelle stratégie de réplication de données qui satisfait les exigences de performances des utilisateurs tout en tenant compte du profit du fournisseur. La création d'une nouvelle réplique de données n'est envisagée que si le temps de réponse dépasse un certain temps de réponse seuil. Par la suite, la réplication doit être profitable pour le fournisseur. Ensuite, cette réplique est placée de telle sorte que la charge de travail soit équilibrée entre les nœuds. Par ailleurs, la stratégie de réplication proposée permet d'ajuster dynamiquement le nombre de répliques. Cela permet une gestion efficace des ressources. L'analyse des résultats montre que la stratégie proposée réduit la consommation de ressources, ce qui améliore le bénéfice du fournisseur tout en satisfaisant les exigences de performance du locataire.

Mots-clés: environnement cloud, bases de données NoSQL; réplication de données; bénéfice du fournisseur; performances.

Abstract

In recent years, data has become more complex and voluminous, due to the rapid change of large scale applications. The management of these data in cloud environments presents an important challenge. In this context, Cloud providers aim to maximize their profits while satisfying tenant requirements, e.g., performance.

The relational database management systems face many obstacles in achieving the performance goal. Therefore, the use of NoSQL databases becomes necessary when dealing with heterogeneous workloads and voluminous data.

In this context, data replication constitutes a well known technique that ensures availability, improves performances and ensures fault tolerance. In this work, we propose a new data replication strategy that satisfies the tenant performance objective while taking into account the provider economic profit. A new replica is created only if the estimated response time is superior to a given response time threshold. Then, the replication should be profitable for the provider. A new replica is placed on a balanced way and the number of replicas is dynamically adjusted.

Result analysis shows that the proposed strategy reduces the resource consumption, which improves the provider profit.

Keywords: Cloud environment, NoSQL databases; data replication; provider profit; performance.

الملخص

في السنوات الأخيرة، أصبحت البيانات أكثر تعقيدا وحجما، نظرا للتغير السريع للتكنولوجيات واسعة النطاق والتطبيقات الجديدة، وإدارة هذه البيانات يمثل تحديا هاما. ويهدف مقدمو الخدمات السحابية إلى مضاعفة أرباحهم مع تلبية متطلبات المستأجر، مثل الأداء. وتواجه نظم إدارة قواعد البيانات الكلاسيكية عقبات عديدة في سبيل تحقيق هذا الهدف. ولذلك، فإن استخدام قواعد بيانات NoSQL، يصبح ضروريا عند التعامل مع أعباء العمل غير المتجانسة والبيانات الضخمة.

في هذا السياق، نقترح إستراتيجية جديدة لاستنساخ البيانات و التي تقوم بوازنة عبء العمل وتقوم بالتعديل الديناميكي لعدد النسخ ، بينما يؤخذ ربح مقدم الخدمة في الاعتبار. ويظهر تحليل النتائج أن الإستراتيجية المقترحة تقلل من استهلاك الموارد، مما يحسن أرباح مقدم الخدمة مع الوفاء بمتطلبات أداء المستأجر. من خلال عدم كسر اتفاقات مستوى الخدمة التي تحد الطرفين.

كلمات البحث: بيئة سحابية، قواعد بيانات NoSQL. تكرار البيانات؛ ربح مقدم الخدمة ؛ الأداء

Table des matières

Dédicace 2

Remerciements 3

Résumé 4

Abstract 5

الملخص 6

Liste des figures 11

Liste des tableaux 13

INTRODUCTION GENERALE 14

1. Contexte du travail et problématique 15

2. Objectifs de la thèse 16

3. Structure de la thèse 16

CHAPITRE 1 : CONCEPTS GENERAUX 18

1.1. Introduction 20

1.2. La réplication de données 21

1.2.1. Définition 21

1.2.2. Avantages et inconvénients de la réplication 21

1.1.3. Stratégie de réplication 21

1.2.3.1. Quand répliquer 22

1.2.3.2. Quoi répliquer 22

1.2.3.3. Où répliquer 22

1.3. Cloud computing 22

1.3.1. Quelques concepts 22

1.3.2. Caractéristiques 23

1.3.3. Principaux modèles de livraisons des services Cloud 26

1.3.3.1. Le modèle Software-as-a-Service (SaaS) 26

1.3.3.2. Le modèle Platform-as-a-Service (PaaS) 26

1.3.3.3. Le modèle Infrastructure-as-a-Service (IaaS) 27

1.3.4. Modèles de déploiement du Cloud 27
1.3.4.1. Cloud Public 28
1.3.4.2. Cloud Privé 28
1.3.4.3. Cloud Hybride 28
1.2.4.4. Cloud Collectif (en communauté) 28
1.4. Le modèle NoSQL 28
1.4.1. Historique et définitions 28
1.4.2. Les types de base de données NoSQL 29
1.4.2.1. Modèle Orienté Clé/Valeur 29
1.4.2.2. Modèle Orienté Document 30
1.4.2.3. Modèle Orienté Colonne 31
1.4.2.4. Modèle Orienté Graphe 32
1.4.3. Analyse comparative des bases de données NoSQL 33
1.4.4. Le modèle NoSQL et Cloud computing 34
1. 5. Conclusion 35
CHAPITRE 2 : ETAT DE L'ART 36
2.1. Introduction 38
2.2. La Réplication de données dans les systèmes cloud 39
2.2.1. Classifications existantes 39
2.2.1.1. Réplication statique vs. réplication dynamique 39
2.2.1.2. Réplication centralisée vs. réplication décentralisée 39
2.2.1.3. Réplication déclenchée par le serveur vs. par Le client 39
2.2.1.4. Classification basée sur les fonctions objectives 40
2.2.1.5. Classification basée sur l'architecture du système 40
2.2.2. La Classification proposée 40
2.2.2.1. Nature de la réplication de données *(statique vs. dynamique)* 41
2.2.2.2. Équilibrage de la charge de travail *(réactif ou proactif)* 43
2.2.2.3. Orientation du bénéfice *(fournisseur vs. client)* 44
2.2.2.4. Fonctions objectives 46

a. Localité temporelle et géographique 46
b. Localité au niveau du réseau 47
c. Modèle de coût économique 47
2.3. Discussion 48
2.4. Introduction aux simulateurs 49
2.5. Facteurs de performance des stratégies de réplication de données 50
2.5.1. Granularité optimale de la réplication de données 50
2.5.2. Latence d'accès 51
2.5.3. Consommation de la bande passante 51
2.5.4. Equilibrage de charge de travail 51
2.5.5. Capacité de stockage 51
2.5.6. Nombre optimal de répliques 51
2.6. La réplication de données dans les systèmes de base des données NoSQL orientées documents : Mongodb 52
2.6.1. Introduction à Mongodb 52
2.6.2. Les travaux connexes 53
2.6.3. Les stratégies de réplication dans Mongodb 54
2.6.3.1. Modèle Maître/Esclave 54
2.6.3.2. Modèle replicaSets 55
2.7. Conclusion 57
CHAPITRE 3 : LA STRATEGIE DE REPLICATION DE DONNEES PROPOSEE 59
3.1. Introduction 60
3.4. La topologie considérée 62
3.5. La stratégie proposée de la réplication de données 63
3.5.1. La décision de la réplication 64
3.5.1.1. Modèle de coût pour le traitement des requêtes 65
- Estimation du temps de réponse d'une requête 65
a. Estimation du temps CPU 66
b. Estimation du temps I/Os 66

c. Estimation du temps de communication 67
3.5.1.2. Modèle économique pour le traitement de requêtes 67
a. Estimation des revenus 67
b. Estimation des dépenses 68
3.5.2. La sélection des données à répliquer 68
3.5.3. Le placement des nouvelle répliques 69
3.5.4. La suppression de répliques 71
3.6. Conclusion 72
CHAPITRE 4 : EVALUATION DES PERFORMANCES 73
4.1. Introduction 74
4.2. Environnement de simulation 75
4.3. Mise en place d'YCSB 75
4.4. Evaluation du mécanisme de réplication de données existant dans MongoDB 76
4.5. Évaluation de la stratégie proposée 79
4.6. Discussion 80
4.7. Conclusion 83
CHAPITRE 5 : ETUDE DE CAS CONTEXTE DE PLANIFICATION DES PROJETS TERRITORIAUX 84
5.1. Introduction 85
5.2. Etude de cas (les opérateurs téléphoniques) 86
5.7. Discussion 91
5.8. Conclusion 93
CONCLUSION GENERALE 94
1. Contribution 95
2. Perspectives 97
Publications et Communications 98
Bibliographie 100

Liste des figures

Figure 1.1. Modèle de service de cloud en couches [17] 27
Figure 1.2. Illustration d'une Base de données Orientée Clé / Valeur [22]. 30
Figure 1.3. Illustration d'une Base de données Orientée Document [22]. 31
Figure 1.4. Illustration d'une Base de données Orientée Colonne. 32
Figure 1.5. Illustration d'une Base de données Orientée Graphe [22]. 33
Figure 2.1. Taxonomie des stratégies de réplication de données dans les systèmes cloud [40] .. 41
Figure 3.1. Modèle Maître/Esclave 54
Figure 3.2. Cas de défaillance de l'esclave. 54
Figure 3.3. Cas de défaillance du serveur maître. 55
Figure 3.4. Modèle replica sets [95] 55
Figure 3.5. Cas de défaillance du serveur maître [95] 56
Figure 3.6. Election d'un nouveau nœud primaire [95] 56
Figure 3.7. Architecture du système cloud adaptée. 62
Figure 3.8. L'architecture de centre de données. 63
Figure 3.9. Algorithme de décision de réplication. 65
Figure 3.10. Données concernées par la réplication. 69
Figure 3.11. Algorithme de placement des nouvelles répliques. 70
Figure 3.12. Algorithme de suppression des répliques. 71
Figure 4.1. Les tests de fonctionnement des dépendances 76
Figure 4.2. Phase de téléchargement et compilation du source YCSB 76
Figure 4.3. Résultats du chargement (load) de la charge de travail A. 77
Figure 4.4. Résultat d'exécution (run) de la charge de travail A. 78
Figure 4.5. Les changements des Résultats de l'exécution des charges de travail. 79
Figure 4.6. Exemple d'un code source utilisé pour l'expérimentation 79
Figure 4.7. Résultats obtenus. 81
Figure 4.8. Nombre de réplications. 82

Figure 5.1. Type des projets urbain....86
Figure 5.2. Caractéristiques d'un opérateur téléphonique....86
Figure 5.3. Transformation des paramétres....87
Figure 5.4: Interface graphique du 'Jetbrains webstorm'....89
Figure 5.5. Insertion des JSON objects avec *JSON GENERATOR* tool....89
Figure 5.6. Format du fichier *JSON* après l'opération de la génération....90
Figure 5.7. Résultat obtenue des tests....90

Liste des tableaux

Tableau 1.1. Comparaison de certaines bases de données NoSQL. 33

Tableau 2.1. Comparaison de certaines stratégies statiques et dynamiques 43

Tableau 2.2. Comparaison des stratégies basées sur l'équilibrage de la charge de travail 44

Tableau 2.3. Comparaison des stratégies centrées sur le client et celles centrées sur le fournisseur 46

Tableau 2.4. Comparaison des stratégies basées sur la fonction objective. 48

Tableau 2.5. Comparaison des simulateurs de cloud. 50

Tableau 3.1. Relation entre la terminologie RDBMS et Mongodb. 53

Tableau 4.1. Spécifications et paramètres de la machine. 75

Tableau 4.2. Description des charges de travail. 77

Tableau 4.3. Résultats de l'exécution des charges de travail. 78

Tableau 5.1. Transformation des paramétres. 87

INTRODUCTION GENERALE

Sommaire

INTRODUCTION GENERALE 14

1. Contexte du travail et problématique 15

2. Objectifs de la thèse 16

3. Structure de la thèse 16

1. Contexte du travail et problématique

Ces dernières années, la popularité croissante des applications, ex, les expériences scientifiques, Internet des objets et les réseaux sociaux, a conduit à la génération de gros volumes de données. La gestion de telles données qui de plus, sont hétérogènes et distribuées à grande échelle, constitue un défi important.

Dans les systèmes traditionnels tels que les systèmes distribués et parallèles, les données sont réparties à travers le monde, alors que la satisfaction des objectifs tels que l'obtention de performances acceptables tout en garantissant une bonne disponibilité de données constituent l'un des objectifs majeurs pour l'utilisateur.

Dans ce contexte, la réplication de données, une technique bien connue, permet notamment: (i) d'augmenter la disponibilité de données, (ii) de réduire les coûts d'accès aux données et (iii) d'assurer une meilleure tolérance aux pannes. Néanmoins, répliquer les données sur tous les nœuds est une solution non réaliste vu qu'elle génère une consommation importante de la bande passante en plus de l'espace limité de stockage. Définir des stratégies de réplication constitue la solution à apporter à ces problématiques.

Les stratégies de réplication de données qui ont été proposées pour les systèmes traditionnels cités précédemment ont pour objectif l'amélioration des performances pour l'utilisateur. Néanmoins, elles sont difficiles à adapter dans les systèmes de cloud. En effet, le fournisseur de cloud a pour but de générer un profit en plus de répondre aux exigences de ces locataires. En effet, les objectifs tels que : (i) satisfaire les attentes de ces locataires en matière de performances sans sacrifier le profit du fournisseur d'un côté, et (ii) la gestion élastiques des ressources avec une tarification suivant le modèle "pay-as-you-go" d'un autre côté, constituent des principes fondamentaux dans les systèmes cloud.

Avec l'augmentation continue des données stockées et analysées, les bases de données relationnelles ont présenté quelques limitations. En conséquence, les bases de données NoSQL ont été développé pour fournir un ensemble de nouvelles fonctionnalités de gestion des données afin de surmonter certaines limites des bases de données relationnelles, en particulier lorsqu'il s'agit de données volumineuses et hétérogènes. Beaucoup des travaux se sont intéressés à l'amélioration des performances dans les systèmes NoSQL. La plupart d'entre eux se sont concentré sur les algorithmes d'auto-sharding et de balance de charge mais peu d'entre eux ont focalisé leur attention sur l'amélioration des stratégies de réplication dans les systèmes NoSQL. Dans ce qui suit, on s'intéresse à la réplication dans les systèmes NoSQL orientés documents . Plus exactement, on s'intéresse à la réplication de données dans le système Mongodb.

Dans cette thèse, nous proposons une nouvelle stratégie de réplication dans les systèmes NoSQL orientés documents *Mongodb*. La stratégie proposée vise à assurer à la fois la satisfaction des exigences du locataire, ex. les performances, et une rentabilité pour le fournisseur du cloud. En se basant sur un modèle de coût, nous estimons le

temps de réponse nécessaire pour l'exécution d'une requête distribuée. La réplication de données n'est envisagée que si le temps de réponse estimé dépasse un seuil fixé auparavant dans le contrat établi entre le fournisseur et son client. Ensuite, cette réplication doit être profitable du point de vue économique pour le fournisseur. Dans ce contexte, nous proposons un modèle économique prenant en compte aussi bien les dépenses et les revenus du fournisseur lors de l'exécution de cette requête. Nous proposons une heuristique pour le placement des nouvelles répliques afin de réduire les temps d'accès à ces nouvelles répliques. De plus, un ajustement dynamique du nombre de répliques est adopté afin de permettre une gestion élastique des ressources.

2. Objectifs de la thèse

Dans le cadre de cette thèse, nous présentons tout d'abord un état de l'art afin de relever les défis de recherches des stratégies existantes de la réplication de données dans les systèmes cloud. Cet état de l'art nous permettra de proposer une nouvelle classification des stratégies de la réplication de données dans ces mêmes systèmes. En effet, nous avons classé les stratégies de réplication en fonction de quatre critères : (a) modèle d'accès utilisateur, (b) équilibrage de la charge de travail, (c) prise en compte du bénéfice du fournisseur; et (d) les fonctions objectives visées. Ensuite, nous décrivons la stratégie de réplication de données proposée qui permet la satisfaction des exigences de l'utilisateur tout en prenant en compte le profit du fournisseur. Dans ce contexte, nous présentons le modèle de cout adopté qui permet l'estimation du temps de réponse d'une requête avant son exécution. Nous présentons également le modèle de cout économique qui permet l'estimation des dépenses et revenues de chaque requête pour le fournisseur. Afin de réduire le temps d'accès aux nouvelles répliques, une heuristique de placement de ces répliques doit être mise au point. Par ailleurs, un algorithme permettant d'ajuster dynamiquement le nombre des répliques est mis au point. Afin de valider la stratégie proposée, une évaluation des performances est faite. Elle permet de vérifier la réduction de la consommation des ressources. Ceci permettra de confirmer que la stratégie de réplication de données proposée permet de satisfaire les exigences de performance du locataire tout en prenant en compte le profit du fournisseur.

3. Structure de la thèse

Dans cette thèse, nous commençons par la description du contexte, des problématiques et les objectifs de notre travail de recherche. Cette thèse est organisée en cinq chapitres. Les chapitres 1 et 2 définissent le cadre général et posent le socle bibliographique de notre travail de recherche tandis que les trois autres chapitres sont consacrés à nos contributions et la validation de notre proposition.

Dans le premier chapitre, nous mettrons l'accent sur la technique de réplication de données, le cloud computing(cloud computing) et le modèle NoSQL. Dans un premier temps, nous présenterons les concepts liés à la technique de réplication de données et les principes d'une stratégie de réplication de données. Puis, nous présenterons les concepts généraux concernant le cloud computing. Ensuite, nous introduisons le modèle

NoSQL, ses différentes catégories**,** ainsi qu'une analyse comparative des différents types de systèmes NoSQL.

Dans le chapitre 2, on parlera de l'état de l'art concernant les stratégies de réplication de données. Une discussion détaillée des stratégies de réplication de données existantes sera fournie par rapport à divers systèmes de gestion de données. Cette discussion aidera à transmettre la justification du système de gestion de données cible qui est considérée dans cette thèse à savoir les systèmes de gestion de base de données dans le cloud. Cette étude nous a permis de proposer une nouvelle classification des stratégies de réplication de données dans les systèmes cloud.

La contribution principale de la thèse est décrite dans le chapitre 3. Dans ce chapitre, on répondra à des questions telles que : quand la stratégie de réplication de données est déclenchée ? Quelles données sont concernées par la réplication de données ? combien de répliques créer lors de chaque réplication ? et où placer les répliques nouvellement créées. Nous nous basons sur un modèle de coût pour l'estimation du temps de réponse d'une requête et sur un modèle de couts économique pour l'estimation des revenus et des dépenses du fournisseur lors de l'exécution d'une requête. En outre, la suppression des répliques inutiles est également décrite dans ce chapitre.

Dans le chapitre 4, Nous validerons la stratégie proposée avec une évaluation expérimentale. Ainsi nous avons comparé la stratégie de réplication existante dans la base de données NoSQL *Mongodb* avec notre stratégie proposée en termes de temps de réponse et de nombre de réplications déclenchées. Nous démontrerons comment la stratégie de réplication de données proposée satisfait simultanément les exigences de performances pour le locataire et de rentabilité du fournisseur.

Dans le chapitre 5, nous avons validé la stratégie proposée avec un exemple de cas d'application, qui offre la possibilité de gérer les infrastructure des opérateurs téléphoniques pour obtenir de meilleures décisions dans le domaine de la planification du territoire urbain **.**

CHAPITRE 1 : CONCEPTS GENERAUX

Sommaire

CHAPITRE 1 : CONCEPTS GENERAUX 18

1.1. Introduction 20

1.2. La réplication de données 21

1.2.1. Définition 21

1.2.2. Avantages et inconvénients de la réplication 21

1.1.3. Stratégie de réplication 21

1.2.3.1. Quand répliquer 22

1.2.3.2. Quoi répliquer 22

1.2.3.3. Où répliquer 22

1.3. Cloud computing 22

1.3.1. Quelques concepts 22

1.3.2. Caractéristiques 23

1.3.3. Principaux modèles de livraisons des services Cloud 26

1.3.3.1. Le modèle Software-as-a-Service (SaaS) 26

1.3.3.2. Le modèle Platform-as-a-Service (PaaS) 26

1.3.3.3. Le modèle Infrastructure-as-a-Service (IaaS) 27

1.3.4. Modèles de déploiement du Cloud 27

1.3.4.1. Cloud Public 28

1.3.4.2. Cloud Privé 28

1.3.4.3. Cloud Hybride 28

1.2.4.4. Cloud Collectif (en communauté) 28

1.4. Le modèle NoSQL 28

1.4.1. Historique et définitions 28

1.4.2. Les types de base de données NoSQL 29

1.4.2.1. Modèle Orienté Clé/Valeur 29

1.4.2.2. Modèle Orienté Document 30
1.4.2.3. Modèle Orienté Colonne 31
1.4.2.4. Modèle Orienté Graphe 32
1.4.3. Analyse comparative des bases de données NoSQL 33
1.4.4. Le modèle NoSQL et Cloud computing 34
1. 5. Conclusion 35

1.1. Introduction

Ces dernières années Il y a eu beaucoup d'évolutions dans les systèmes de gestion de base de données. Le besoin de base de données évolutives s'est fait encore plus ressentir avec la croissance des données dans le monde du Web. Ceci a provoqué l'apparition des bases de données NoSQL avec son évolutivité élevée et son modèle facile à programmer. Par conséquent, le traitement des données sur Internet est devenu rentable. De plus, la demande en base de données NoSQL a augmenté car la plupart des applications cloud exigent une haute disponibilité, une vitesse, des options de basculement et une tolérance aux pannes qu'une base de données relationnelle traditionnelle ne parvient pas à offrir aux applications Web modernes. Pour de nombreuses applications scientifiques ayant recours aux systèmes à large échelle, les quantités de données nécessaires aux traitements de ces applications sont très importantes. D'un autre côté, de telles quantités de données impliqueraient de très grands délais de transfert si seule une copie de ces données était stockée. En conséquence, le temps de réponse et la disponibilité des données deviennent les défis principaux à adresser. Afin de répondre à ces défis, une technique importante est de répliquer les données dans différents sites, de sorte qu'un utilisateur puisse accéder à une copie telle que la qualité de service (par exemple ; ainsi les performances) est améliorée. De plus, dans les environnements à grande échelle tel que les clouds, le profit du fournisseur doit être pris en compte ce qui constitue une différence majeure avec les stratégies de réplication déjà proposées dans les systèmes traditionnels. Ce chapitre vise à faire une présentation des notions clés permettant de définir et comprendre le cadre général de notre travail de recherche à savoir la technique de réplication de données, le cloud computing, et le modèle NoSQL.

1.2. La réplication de données

1.2.1. Définition

La réplication de données est une technique bien connue utilisée dans les systèmes distribués, et qui permet d'améliorer la fiabilité, la tolérance aux pannes, l'accessibilité et augmenter la disponibilité des données.

La réplication consiste à créer des copies d'une donnée et de les stocker dans des endroits différents, en mettant en œuvre un processus de création et de placement des copies d'entité logiciel. La phase de création consiste à reproduire la structure et l'état des entités répliquées, tandis que la phase de placement consiste à choisir, en fonction des objectifs de la réplication, le bon emplacement de cette nouvelle reproduction. Les lectures sont exécutées sur le site disposant de la copie la plus proche du client, ce qui diminue le nombre de transferts de données. Quand à la disponibilité des données, la réplication permet de ne plus dépendre d'un site central unique et utiliser une copie sur un site lorsqu'elle est indisponible sur un autre [1], [2]. De plus, la réplication permet de distribuer la charge de travail sur différents nœuds possédant une réplique.

1.2.2. Avantages et inconvénients de la réplication

La réplication présente des avantages différents selon le type de réplication et les options choisies. Mais, l'intérêt général de la réplication est la disponibilité des données à tout moment et en tout lieu. Le recours à la technique de réplication procure certains avantages que nous pouvons énumérer comme suit :

- Disponibilité de données : les données sont disponibles localement même en l'absence de toute connexion à un serveur central. En conséquence, l'utilisateur n'est pas coupé de ses données en cas de défaillance d'une connexion réseau,
- L'amélioration du temps de réponse des requêtes d'interrogation du fait d'une meilleure bande passante suite à la réplication de données distantes.
- Amélioration de la fiabilité et de la sureté de fonctionnement,
- La redondance permet une meilleure protection contre la corruption de fichiers.

Néanmoins, la réplication peut présenter quelques problèmes tels que :

- le coût de mise à jour : la modification de la copie originale engendre la mise à jour de l'ensemble des copies. le coût de mise à jour dépend donc du nombre de copies. De plus, la propagation des mises à jour nécessite généralement un certain temps ce qui entraine une période de repos (pas de services). une requête qui arrive pendant cette période sera donc retardée.

1.1.3. Stratégie de réplication

Répliquer toutes les données partout ne constitue pas une bonne solution à cause des contraintes de stockage et de bande passante. Une stratégie de réplication de données devient donc nécessaire pour toute réplication de données. Le but de toute stratégie de

réplication de données est de déterminer quelles données doivent être répliquées, quand une réplique doit être créée / supprimée, combien de répliques doivent être créées et où placer une nouvelle réplique [3]. Dans les environnements cloud, un autre aspect est pris en compte; une stratégie de réplication proposée devrait également être rentable pour le fournisseur, tandis que les objectifs des locataires sont satisfaits.

1.2.3.1. Quand répliquer

Ceci renseigne sur le moment de réplication. Dans la stratégie proposée dans cette thèse, la décision de la réplication dépend de la vérification de deux conditions: (i) un temps de réponse est supérieur à un seuil de temps de réponse et (b) par la suite, une réplication devrait être profitable au fournisseur.

1.2.3.2. Quoi répliquer

Une stratégie de réplication doit pouvoir répondre à la question 'quoi répliquer?' les données répliquées sont généralement des fichiers ou des objets. Ces objets peuvent être composés d'un ensemble de fichiers distribués (appelés collections dans le système MongoDB). Selon les stratégies de réplication, les données à répliquer, peuvent être les plus usuelles ou encore celles auxquelles on accède le plus fréquemment.

1.2.3.3. Où répliquer

Une stratégie de réplication doit également répondre à la question 'où répliquer?' les stratégies de placement de répliques doivent tenir compte du fait que les sites potentiels ne possèdent pas des répliques de cette même donnée. De plus, ces sitesdoivent posséder l'espace de stockage suffisant et sont à une distance raisonnable en termes de temps de transfert.

1.3. Cloud computing

1.3.1. Quelques concepts

Afin de discerner le Cloud computing, nous allons présenter un ensemble de définitions qui éclaircissent chacune des facettes du Cloud :

Selon [4], le Cloud computing est un type de calcul qui offre un accès simple et sur demande, d'un ensemble de ressources informatiques hautement élastiques. Ces ressources sont fournies comme service sur Internet, et sont maintenues grâce à des séries d'innovations et d'entretiens. Le Cloud permet aux utilisateurs de minimiser le cout d'utilisation, en se libérant de se préoccuper de la façon dont les ressources son gérées et où elles se trouvent.

Selon [5], le Cloud est un vaste regroupement de ressources virtuelles, facilement accessibles et utilisables. Ces ressources peuvent être dynamiquement reconfigurées pour s'adapter à une charge variable, ce qui permet une utilisation optimale des ressources. Ce regroupement de ressources est généralement exploité par un modèle pay-per-use, dans lequel des garanties sont offertes par le fournisseur de services au moyen des contrats de niveau de services.

Selon [6], cloud computing est un modèle qui permet d'accéder au réseau de façon ubiquitaire, facile et sur demande. Ce modèle offre la possibilité de partager et configurer les ressources Cloud (réseaux, serveurs, stockage, applications et services) qui peuvent être rapidement approvisionnées et libérées avec un effort de gestion et des interactions minimales.

En analysant ces trois définitions, on constate que Le cloud computing n'est qu'un ensemble d'outils de virtualisation et de gestion de ressources, basé sur de multiples machines. Ces machines sont regroupées en un seul système dont les ressources sont éventuellement distribuées. Ceci doit assurer et satisfaire un ensemble de caractéristiques afin de faciliter l'accès à un large spectre de ressources.

1.3.2. Caractéristiques

Dans cette section, nous verrons les principaux caractéristiques du Cloud Computing.

- ***Auto-guérison***

Tout système Cloud doit contenir une ou plusieurs copies de chaque application déployée en lui, de telle façon qu'en cas de dysfonctionnement de l'application en cours, sa copie vient la remplacer en prenant l'état actuel de l'application en échec. Les applications en copies doivent être maintenues et mises à jour à chaque fois que l'application en cours est modifiée [7].

- ***Multi locations (Multi-Tenancy)***

Sur le Cloud, une même application peut être utilisée par plusieurs clients en même temps, en préservant la sécurité et les données privées de chaque client. Cela est possible en utilisant des outils de virtualisation qui permettent de partager un serveur sur plusieurs utilisateurs [8].

- ***Evolutivité linéaire***

Un système Cloud a la faculté de découper les principales taches du système en un ensemble de petits morceaux, et de les distribuer par la suite sur l'infrastructure virtuelle du Cloud [9].

- ***Offre de services***

Tout système Cloud est basé sur un ensemble de services indépendants les uns des autres. Ce qui donne de la puissance à la technologie Cloud c'est le rassemblement de ces services en une seule unité afin de présenter un service Cloud qui couvre tous les niveaux sur lesquels une application est basée (Niveau virtuel, niveau logiciel, une interface facile à utiliser …. Etc.) [10]

- ***Virtualisation***

Un système Cloud est un système complètement virtuel et indépendant de la couche physique sur laquelle les ressources et applications sont déployées [12].

- ***Flexibilité***

Les services Cloud sont destinés à supporter aussi bien les lourdes que les petites charges de travail en augmentant ou réduisant automatiquement les ressources utilisées selon la taille des taches à traiter [13].

- ***Un contrat SLA (Service Level Agrement)***

Avec les services Cloud, un client peut négocier le niveau de service qui lui convient moyennant pour cela un coût à payer. Dans le cas où les ressources Cloud sont en surcharge, le système crée d'autres entités d'applications Cloud en utilisant les outils de virtualisation disponibles afin de respecter les termes du contrat SLA [11].

Le SLA est un contrat qui quantifie le niveau de service minimal pour une prestation qu'un fournisseur s'engage à délivrer à son client. Il peut être soit une partie d'un contrat informatique, soit une annexe à un contrat informatique, soit une annexe à des conditions générales. En français il peut être appelé sous différents noms comme « Accord de niveau de service », « Contrat de service », « Convention de service », etc…

Les principaux buts d'un SLA sont de définir :

- les besoins d'un client pour pouvoir les exprimer clairement d'une manière compréhensible par chacune des parties (fournisseur et client),
- les critères dévaluation ainsi que les moyens de mesure avec lesquels on pourra évaluer la qualité de la prestation fournie.

Les autres buts sont entre autres :

- d'établir une relation de confiance, voir de partenariat entre les parties,
- d'éliminer les attentes irréalistes ou trop chères (rapport prix/prestation).

La rédaction d'un SLA varie en fonction de la prestation qui va être fournie, c'est à-dire selon le champ d'application. En effet un SLA "Réseau" sera différent d'un SLA "Hébergement" qui lui, sera différent d'un SLA "Application". Chacun comportera un certain nombre de points spécifiques liés au domaine qu'il va devoir couvrir. Néanmoins tous les types de SLA doivent couvrir un certain nombre d'aspects afin d'être complets.

La liste ci-dessous va traiter un certain nombre de points qui sont à prendre en compte pour qu'un SLA soit le plus complet possible :

- Le lien et la priorité du SLA par rapport à d'autres contrats.
- La liste et la définition du ou des services.
- L'obligation d'information et de conseil.
- L'utilisation du/ des services par le client.
- Les critères de mesure permettant de définir le niveau de qualité.
- Le prix.
- Le début, la fin, le renouvellement et la résiliation du SLA.
- La modification du service, donc du SLA.

- La récupération.
- Les responsabilités et obligations (du fournisseur et du client)qui présentent les objectifs en tant qu'objectif de niveau de services (SLO), associés à une métrique définie par l'utilisateur qui doit être satisfaite par le fournisseur, par exemple: le temps de réponse, disponibilité) et (ii) garantie.
- La sécurité.
- La maintenance.
- Les relations avec des tiers.
- Les audits.
- Les plaintes et les pénalités : valeur monétaire versée par le fournisseur à l'utilisateur pour ne pas avoir satisfait SLO (en cas de violation de l'accord)
- Le droit applicable.

Exemple 1 : de niveau de service (disponibilité)

Service disponible annuellement à 99,9% 7j/7, 24h/24 (typiquement site web).
Dans ce cas, 24 heures de fonctionnement sur 30 jours par mois pendant 12 mois, donnent 24*12*30 = 8'640 heures. 99.9% de 8'640 heures = 8631.4 heures. Ceci veut dire que le service, tout en respectant le SLA, peut être indisponible chaque année pendant 8.6 heures.

Il est important de savoir que dans ce cas, le fournisseur va demander des plages de maintenance qui seront indispensables au bon fonctionnement du service. Ces plages ne pourront pas être considérées comme une indisponibilité du service.

Exemple 2: de niveau de service (disponibilité)

Il est intéressant de savoir que les fournisseurs qui proposent un service annuel à 99.99%, 24h/24, 7j/7, disposent d'un temps d'indisponibilité annuel de 52 minutes !

Exemple 3 : lieux dispersés

Un fournisseur délivre une prestation avec du matériel chez le client. Si le client se trouve dispersé sur différents sites, le fournisseur devra pouvoir être en mesure d'assurer la disponibilité définie. En effet s'il doit intervenir sur site pour le remplacement de matériel défectueux, il va falloir qu'il prenne en compte les temps de déplacement et les frais liés.

Exemple 4: catégorisation de données (sécurité)

- Publique: accessibles à tous
- Interne : à usage interne de l'entreprise
- Confidentielle: à usage limité à un certain nombre de personnes définies
- Secrète: à usage très limité à 2 ou 3 personnes de la direction.

1.3.3. Principaux modèles de livraisons des services Cloud

Dans cette section, nous verrons les principaux modèles de livraison des services Cloud.

1.3.3.1. Le modèle Software-as-a-Service (SaaS)

Avant l'apparition des technologies et modèles Cloud computing, les clients achètent des logiciels en mode téléchargement ou livraison à domicile, ce qui met le client mal à l'aise. L'avènement du modèle de déploiement Software-as-a-Service, a complètement bouleversé l'industrie logicielle dans le monde, avec ce modele le client alloue un logiciel à installer, stocker et à exécuter sur le Cloud [14]. Par rapport au modèle précédent, le modèle Software-as-a-Service représente des facilités considérables pour la navigation et l'exploitation des ressources physiques et logiques sur le Net [14]:

- La mise à jour du logiciel est automatique, et c'est l'entreprise Cloud qui s'occupe de cela.
- La licence de l'utilisation d'un logiciel sur le Cloud est toujours associée à un contrat de niveau de service, qui assure au client un certain degré de qualité de service et clarifie les droits et devoirs du client vis-à-vis de l'entreprise qui offre le logiciel sur le Cloud.
- Le Client ne paye qu'un abonnement annuel ou mensuel ou ne paye que ce qu'il a utilisé, selon le système de payement du fournisseur de services Cloud.
- Puisque le logiciel alloué par le client est exécuté sur le Cloud, ce dernier n'a pas à se soucier de l'infrastructure physique (Compatibilité avec le côté matériel) ni le côté logique (compatibilité avec le système d'exploitation).
- Enfin, les clients payent généralement une charge supplémentaire pour que le fournisseur de service Cloud s'occupe de la sécurisation de leurs données.

1.3.3.2. Le modèle Platform-as-a-Service (PaaS)

Ce modèle représente une variante du modèle SaaS, sauf qu'au lieu d'offrir une application comme service, il offre un environnement de développement d'applications comme service, ce modèle permet au développeur d'utiliser et réutiliser un large spectre d'outils, de composants logiciels, et de blocs de codes, ce qui lui permet de développer rapidement son application. Avec ce modèle, les fournisseurs des services Cloud offrent un ensemble d'outils et d'environnement de: développement, exécution, déploiement, distribution et de payement des applications créées au niveau de la plateforme.

Le fournisseur de services d'une plateforme de développement est aussi destiné à produire, proposer et développer continuellement des standards ainsi que des outils de développement pour faciliter la tâche aux programmeurs et aux développeurs d'applications Cloud. Une plateforme de développement sur le Cloud permet de réduire considérablement les couts de déploiement des applications avec une grande rapidité de propagation vu que les fournisseurs des services Cloud placent assez de canaux

de communications et de liaisons entre l'application développée et l'infrastructure de déploiement [15].

1.3.3.3. Le modèle Infrastructure-as-a-Service (IaaS)

Les fournisseurs d'infrastructures classiques et celles basées sur le Cloud permettent d'offrir des ressources matérielles et logicielles qui permettent d'héberger et d'offrir des services sur le net. La différence cruciale entre ces deux modèles est qu'un fournisseur IaaS sur le Cloud met en œuvre un système de payement basé sur le taux de consommation des ressources physiques et virtuelles, qui peuvent évoluer selon la demande du consommateur.

Avec le modèle IaaS, le fournisseur peut contrôler l'augmentation, la réduction et l'accès aux ressources de l'infrastructure. Ce contrôle facile et efficace des capacités de l'infrastructure Cloud permet au fournisseur d'héberger des applications web, ou bien, d'aller plus loin en hébergeant des software-as-a-service, des Platform-as-a-service ou des outils de support pour l'amélioration des applications hébergées, ce qui permet aux entreprises d'externaliser leurs ressources et de se débarrasser de la gestion du personnel et des ressources matérielles et logicielles nécessaires à l'hébergement de leurs applications [16].

Les modèles de livraisons des services Cloud peuvent être représentés suivant la figure 1.1 [17].

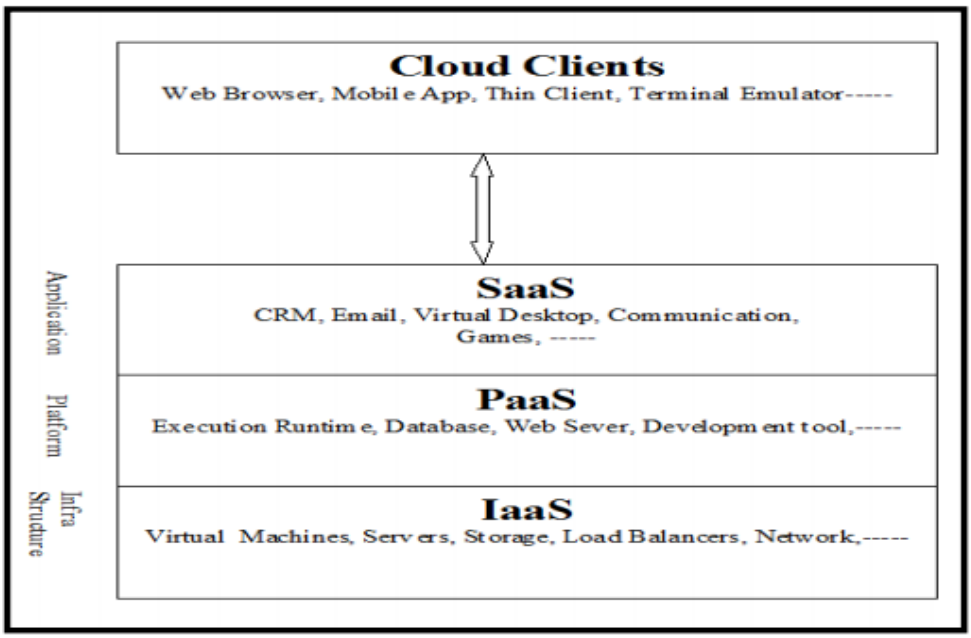


Figure 1.1. Modèle de service de cloud en couches [17]

1.3.4. Modèles de déploiement du Cloud

En plus des modèles de livraison qui permettent de concrétiser les services Cloud, on trouve un ensemble de modèles de déploiement de services basés Cloud computing. Ces modèles permettent de définir le degré d'accès de l'utilisateur final

aux fournisseurs de services Cloud. Ces modèles sont divisés en quatre grandes catégories.

1.3.4.1. Cloud Public

Le Cloud public, appelé aussi Cloud externe [18], représente le Cloud traditionnel utilisé par la majorité des clients sur Internet. Avec un Cloud public, les ressources sont auto-approvisionnées dynamiquement via des applications ou des services web, possédés par des fournisseurs de ressources Cloud qui partagent ces derniers, et qui produisent des factures pour cela.

1.3.4.2. Cloud Privé

Le Cloud privé, appelé aussi Cloud interne, est disponible sur des réseaux privés et qui ne sont accessibles que par un ensemble de clients qui ont l'autorisation d'y accéder [19]. Principalement, ce modèle permet d'avoir plus de contrôles sur la sécurité des données et des applications sur le Cloud. Aussi, il facilite la gestion des ressources Cloud par le propriétaire de ces derniers et augmente la fiabilité du système Cloud.

Dun autre côté, les organisations qui ciblent ce genre de modèle doivent eux-mêmes posséder, gérer, et réduire le personnel nécessaire à la construction et l'utilisation des ressources Cloud, ce qui nécessite de grands investissements en termes d'argent et de personnel pour mettre en œuvre ce type Cloud.

1.3.4.3. Cloud Hybride

Avec le modèle hybride, les organisations peuvent associer le Cloud privé avec le public, de cette façon, ces organisations peuvent déployer leurs principales et sensibles applications sur leur Cloud privé et déployer le reste sur le Cloud public, ce qui peut réduire considérablement les coûts de construction et de management des ressources sur le Cloud.

1.2.4.4. Cloud Collectif (en communauté)

Il s'agit d'un genre de Cloud où les ressources sont partagées par plusieurs organisations. Cette communauté d'organisation peut partager les tâches de gestion de ces ressources, comme la sécurisation des données, le déploiement d'application, l'authentification … etc. ces ressources, qui peuvent être installées dans ou hors sites (les sites des organisations de la communauté), peuvent être gérées par une partie tierce comme par une des organisations faisant partie de la communauté [20].

1.4. Le modèle NoSQL

1.4.1. Historique et définitions

Le modèle relationnel a montré ses limites pour la gestion des données massives du Web qui a connu une révolution avec l'avènement des sites web à fort trafic de données tels que Facebook, Amazon et LinkedIn.

Ces grands acteurs du web ont été rapidement limités par les dits systèmes pour les raisons suivantes:

- Les gros volumes de données.
- Les montées en charge.
- L'hétérogénéité des données.

N'ayant pas trouvé de solution sur le marché répondant à ces problèmes, ils décidèrent de développer chacun, en interne, leurs propres systèmes de gestion de données (SGBD). Ces produits SGBD développés de manière distincte sont connus sous le nom de SGBD NoSQL (Not only SQL) ou de SGBD non relationnelle. Les besoins en performances, lors des traitements de gros volumes de données ainsi que d'augmentation de trafic, ne touchent pas seulement les fameux sites mentionnés ci-dessus, mais aussi de nombreuses entreprises de tous types d'industries confondues. C'est de ce constat qu'est né le mouvement NoSQL [21].

En 1998, apparait pour la première fois dans le monde, le terme NoSQL, employé par Carlo Strozzi pour nommer son SGBD relationnel Open Source léger qui n'utilisait pas le langage SQL. Ironiquement, la trouvaille de M. Strozzi n'a rien à voir avec la mouvance NoSQL que l'on connaît aujourd'hui, vu que son SGBD est de type relationnel. En effet, c'est en 2009, lors d'un rassemblement de la communauté des développeurs des SGBD non-relationnels, que le terme NoSQL a été mis au goût du jour pour englober tous les SGBD de type non-relationnel [21].

NoSQL est une combinaison de deux mots: No et SQL. Ceci pourrait être mal interprétée car l'on pourrait penser que cela signifie la fin du langage SQL et qu'on ne devrait donc plus l'utiliser. En fait, le « No » est un acronyme qui signifie « Not only », c'est-à-dire en français, « non seulement ou pas seulement ». C'est donc une manière de dire qu'il y a autre chose que les bases de données relationnelles [21].

NoSQL est un mouvement récent et simple. Il vise à proposer des alternatives aux bases de données relationnelles pour coller aux nouvelles tendances et architectures du moment, notamment le cloud computing. Les systèmes NoSQL sont caractérisés par une haute disponibilité et un partitionnement horizontal des données, au détriment de la cohérence, alors que les bases de données relationnelles actuelles sont basées sur les propriétés ACID (Atomicité, Consistance ou Cohérence, Isolation et Durabilité) [22].

1.4.2. Les types de base de données NoSQL

Les bases de données gérées par un système NoSQL ne sont plus fondées sur l'architecture classique des bases relationnelles. On distingue Quatre catégories [23].

1.4.2.1. Modèle Orienté Clé/Valeur

Les bases de données type clé / valeur s'articulent sur une architecture très basique. On peut les apparenter à une sorte de HashMap, c'est-à-dire qu'une valeur, un nombre ou du texte est stocké grâce à une clé, qui sera le seul moyen d'y accéder. Leurs

fonctionnalités sont tout autant basiques, car elles ne contiennent que les commandes élémentaires du CRUD (Create, Read, Update, Delete).

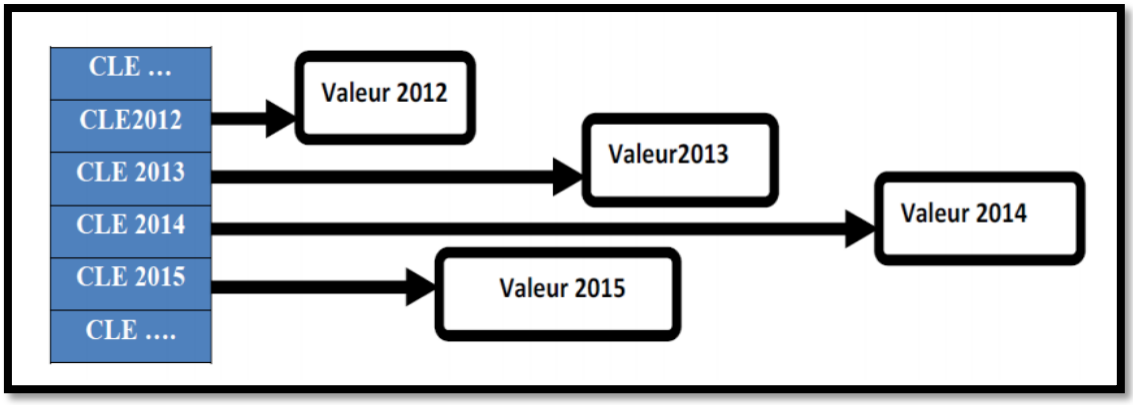


Figure 1.2. Illustration d'une Base de données Orientée Clé / Valeur [22].

Cette approche permet de conserver des performances élevées en lecture/écriture, car de simples accès disques sont effectués pour accéder aux données. Cela permet aux données totalement hétérogènes entre elles d'être stockées. Comme les valeurs stockées sont du type simple (un entier, du texte, un tableau), c'est au développeur de gérer la façon dont elles sont stockées afin de pouvoir les récupérer. base de type clé/valeur les plus utilisées sont Redis et Riak [23].

1.4.2.2. Modèle Orienté Document

Ces bases de données sont une évolution des bases de données de type clé -valeur. Ici les clés ne sont plus associées à des valeurs sous forme de bloc binaire mais à un document dont le format n'est pas imposé. Il peut être de plusieurs types différents comme par exemple du JSON ou du XML, pour autant que la base de données soit en mesure de manipuler le format choisi afin de permettre des traitements sur les documents. Dans le cas contraire, cela équivaudrait à une base de données clé -valeur. Bien que les documents soient structurés, ce type de base est appelée « schemaless ». Il n'est donc pas nécessaire de définir les champs d'un document [23]. On retrouve principalement MongoDB et Couch Base comme solutions basées sur le concept de base documentaire.

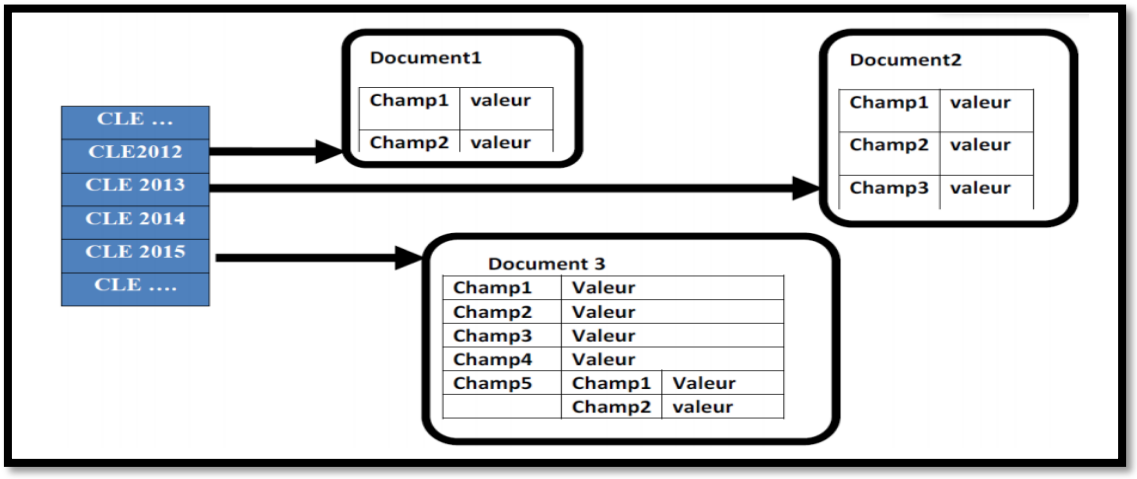


Figure 1.3. Illustration d'une Base de données Orientée Document [22].

1.4.2.3. Modèle Orienté Colonne

Les bases de données orientées colonne ont été conçues par les géants du web afin de faire face à la gestion et au traitement de gros volumes de données. Elles intègrent souvent un système de requêtes minimalistes proche du SQL.

Le principe d'une base de données colonne consiste dans son stockage par colonne et non par ligne, c'est-à-dire que dans une base de données orientée ligne (SGBD classique) on stocke les données de façon à favoriser les lignes en regroupant toutes les colonnes d'une même ligne ensemble. Les bases de données orientées colonne quant à elles vont stocker les données de façon à ce que toutes les données d'une même colonne soient stockées ensemble. Ces bases peuvent évoluer avec le temps, que ce soit en nombre de lignes ou en nombre de colonnes. Autrement dit, et contrairement à une base de données relationnelle où les colonnes sont statiques et présentes pour chaque ligne, celles des bases de données orientées colonnes sont dites dynamiques et présentes donc uniquement en cas de nécessité [23]. Les bases les plus connues se basant sur ce concept sont HBase et Cassandra.

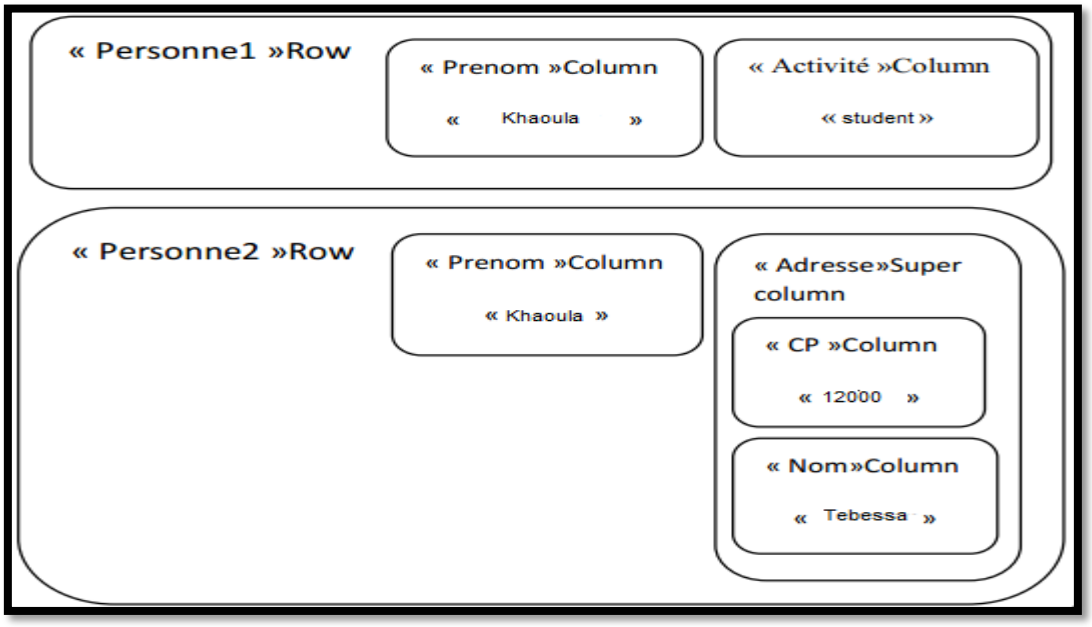


Figure 1.4. Illustration d'une Base de données Orientée Colonne.

1.4.2.4. Modèle Orienté Graphe

Bien que les bases de données de type clé-valeur, colonne, ou document tirent leurs avantages de la performance du traitement de données, les bases de données orientées graphe permettent de résoudre des problèmes très complexes qu'une base de données relationnelle serait incapable de faire. Les réseaux sociaux (Facebook, Twitter, etc), où des millions d'utilisateurs sont reliés de différentes manières, constituent un bon exemple : amis, fans, famille etc. Le défi ici n'est pas le nombre d'éléments à gérer, mais le nombre de relations qu'il peut y avoir entre tous ces éléments. En effet, il y a potentiellement n^2 relations à stocker pour n éléments.

L'approche par graphes devient donc inévitable pour les réseaux sociaux tels que Facebook ou Twitter [23]. Comme l'indique leur nom, ces bases de données reposent sur la théorie des graphes, avec trois éléments à retenir [23] :

- Un objet (dans le contexte de Facebook nous allons dire que c'est un utilisateur) sera appelé un nœud.
- Deux objets peuvent être reliés entre eux (comme une relation d'amitié).
- Chaque objet peut avoir un certain nombre d'attributs (statut social, prénom, nom etc.) , La principale solution est Neo4J.

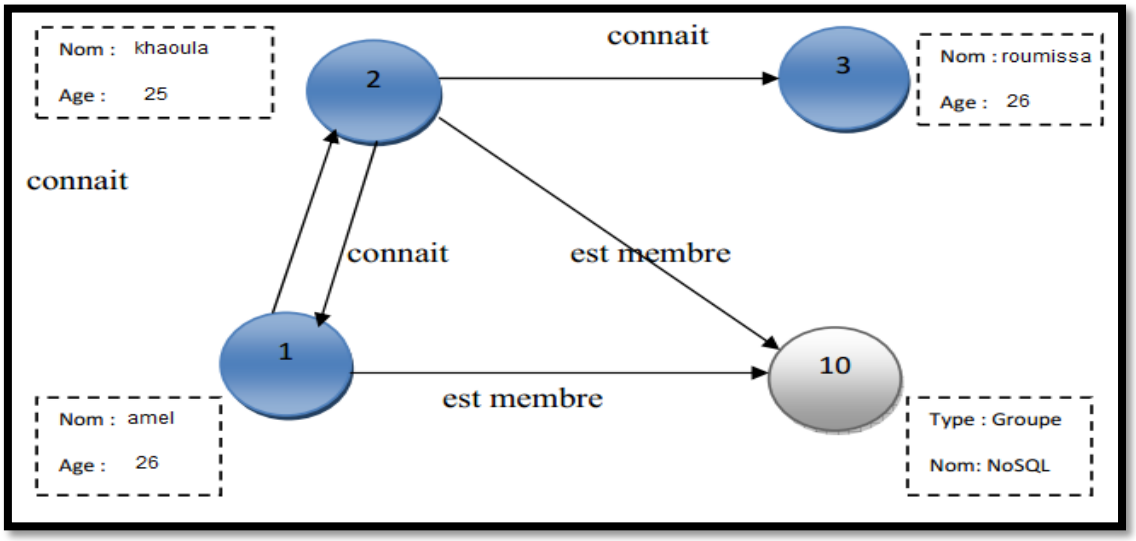


Figure 1.5. Illustration d'une Base de données Orientée Graphe[22].

1.4.3. Analyse comparative des bases de données NoSQL

Les systèmes NoSQL sont généralement utilisés dans des bases de données très volumineuses, particulièrement sujettes à des problèmes de performances dus aux limitations du modèle relationnel [24], centrées sur des classes de problèmes particulières: les données stockées pour cibler des cas d'utilisation, comme les relations et l'agrégation de données, ou simplement pour simplifier l'idée d'une base de données vers quelque chose qui stocke une valeur.

Le tableau 1.1 représente les principales différences entre les 4 catégories de bases de données NoSQL.

	NoSQL Databases			
	Key-value store	***Document store***	***Column store***	***Graph store***
Features	**Redis**	**MongoDB**	**Cassandra**	**Neo4j**
Programming Language	**C++ C**	**C++**	**JAVA**	**JAVA**
Operating System	**Linux, Mac OS, Windows**	**Multi-platform software**	**Multi-platform software**	**Multi-platform software**
Horizontal Scalable	**OUI**	**OUI**	**OUI**	**NO**
Replication Mode	**Master-Slave replication**	**Master-Slave replication**	**Master-Slave replication**	**--**
Sharding	**No**	**OUI**	**OUI**	**OUI**
MapReduce	**NO**	**OUI**	**OUI**	**NO**
Data Storage	**Volatil memory File system**	**Volatil memory File system**	**Volatil memory File system**	**Volatil memory File system**

Tableau 1.1. Comparaison de certaines bases de données NoSQL.

Après avoir analyser les différentes solutions Nosql existantes, nous pouvons extraire les critères de choix en fonction des besoins de l'utilisateur et du type d'opérations qui sont exécutées sur des ensembles de données. Le but de chaque

entreprise est de garantir les performances du système pour ses utilisateurs. Cela dépend du contexte d'utilisation. Parmi les bases de données NoSQL comparées, il est clair qu'il y a celles optimisées pour les opérations de lecture et d'autres, pour les opérations de mises à jour tandis que d'autres pour les opérations d'analyse:

- Pour les opérations en lecture seule, nous évoluons vers une architecture orientée document telle que: Mongodb
- Pour les opérations de mise à jour et d'analyse, il est très intéressant d'adopter des architectures orientées colonnes telles que Cassandra, car elles se sont révélées performantes.
- Pour les architectures à valeur-clé, par rapport aux autres, beaucoup d'efforts restent à faire pour améliorer leurs performances.

Parmi les solutions existantes de bases de données NoSQL il n'y en a pas qui satisfait toutes les exigences mais cela dépend du contexte d'utilisation et de la variation des conditions de déploiement.

1.4.4. Le modèle NoSQL et Cloud computing

Avec des avancées diverses dans le domaine de l'informatique, l'évolutivité, l'utilisation des ressources et les économies d'énergie se voient accorder des priorités plus élevées. Ainsi, afin de tirer parti de la technologie de le cloud computing, les fournisseurs SQL ont mis au point deux approches, le sharding manuel et la mise en cache. Mais cela ne suffit pas pour faire face aux applications web modernes du qu'elles doivent changer en fonction des besoins futurs. Donc, avec les bases de données SQL traditionnelles, l'agilité ne peut pas être atteinte. D'un autre côté, la base de données NOSQL et le cloud computing vont de pair avec les bases de données SQL.

Le principal avantage des bases de données NoSQL est l'agilité. La demande de NoSQL est en pleine croissance car la plupart des applications cloud nécessitent une haute disponibilité, une vitesse, des options de basculement, une tolérance aux pannes et une cohérence qu'une base de données relationnelle traditionnelle n'offre pas aux applications Web modernes [25].

Pour le classement des bases de données NoSQL, MongoDB reste le leader et la solution open source la plus populaire. Cette base de données NoSQL orientée-document tire sa popularité de sa facilité et sa capacité de supporter la montée en charge des applications les plus exigeantes.

1. 5. Conclusion

Dans ce chapitre, nous avons passé en revue les notions clés liées à notre travail de recherche incluant le modèle NoSQL, les principaux concepts et caractéristiques du cloud computing et l'introduction à la technique de réplication de données. Concernant la réplication de données, nous avons introduit les stratégies de réplication de données. Pour le modèle NoSQL, nous avons fait une étude comparative des différents systèmes existants sur le marché actuellement. Nous avons pu extraire les critères de choix en fonction des besoins de l'utilisateur et du type d'opérations qui sont exécutées. Dans le chapitre qui suit, nous aborderons les différentes stratégies de réplication de données ainsi que les travaux existants dans la littérature. Ces travaux reflètent un socle théorique pour la proposition d'une nouvelle classification des stratégies existantes de réplication de données dans les systèmes cloud.

CHAPITRE 2 : ETAT DE L'ART

Sommaire

CHAPITRE 2 : ETAT DE L'ART 36

2.1. Introduction 38

2.2. La Réplication de données dans les systèmes cloud 39

2.2.1. Classifications existantes 39

2.2.1.1. Réplication statique vs. réplication dynamique 39

2.2.1.2. Réplication centralisée vs. réplication décentralisée 39

2.2.1.3. Réplication déclenchée par le serveur vs. par Le client 39

2.2.1.4. Classification basée sur les fonctions objectives 40

2.2.1.5. Classification basée sur l'architecture du système 40

2.2.2. La Classification proposée 40

2.2.2.1. Nature de la réplication de données *(statique vs. dynamique)* 41

2.2.2.2. Équilibrage de la charge de travail *(réactif ou proactif)* 43

2.2.2.3. Orientation du bénéfice *(fournisseur vs. client)* 44

2.2.2.4. Fonctions objectives 46

a. Localité temporelle et géographique 46

b. Localité au niveau du réseau 47

c. Modèle de coût économique 47

2.3. Discussion 48

2.4. Introduction aux simulateurs 49

2.5. Facteurs de performance des stratégies de réplication de données 50

2.5.1. Granularité optimale de la réplication de données 50

2.5.2. Latence d'accès 51

2.5.3. Consommation de la bande passante 51

2.5.4. Equilibrage de charge de travail 51

2.5.5. Capacité de stockage 51

2.5.6. Nombre optimal de répliques 51

2.6. La réplication de données dans les systèmes de base des données NoSQL orientées documents : Mongodb 52

2.6.1. Introduction à Mongodb 52

2.6.2. Les travaux connexes 53

2.6.3. Les stratégies de réplication dans Mongodb 54

2.6.3.1. Modèle Maître/Esclave 54

2.6.3.2. Modèle replicaSets 55

2.7. Conclusion 57

2.1. Introduction

Ce chapitre présente un état de l'art sur les stratégies existantes de réplication de données dans les systèmes Cloud. La réplication de données permet d'assurer une disponibilité de données, améliore les performances et la tolérance aux pannes, et réduit le trafic de données sur le réseau. Elle est fréquemment utilisée dans : (i) Les systèmes de gestion de bases des données [26], (ii) les systèmes parallèles et distribués ([27];[28]), et (iii) les systèmes à grande échelle, y compris les systèmes P2P [29] et les grilles des données ([30];[31]). Chaque stratégie de réplication des données doit répondre aux questions suivantes [32] :

- Quelles données devraient être répliquées ?
- Quand les données doivent-elles être répliquées ?
- Où devrait-on placer de nouvelles répliques?
- Combien de répliques doit-on créer ?

La plupart des travaux dans la littérature ont classifié les stratégies de réplication de données selon les critères suivants : (i) Réplication statique vs. Réplication dynamique ([33];[34]), (ii) Réplication centralisée vs. Réplication décentralisée ([35];[36];[37]), (iii) Réplication déclenchée par le serveur vs. Par le client ([38];[39]), (iv) Classification basée sur les fonctions objectives [3], et (v) Classification basée sur l'architecture du système [31].

Cependant, les stratégies existantes de réplication de données proposées pour les systèmes classiques, tels que les grilles, ne sont pas adaptées aux systèmes Cloud. Elles visent à obtenir de meilleures performances sans tenir compte des profits du fournisseur de cloud. Néanmoins, la création d'autant de répliques en cloud peut ne pas être économiquement faisable. Par conséquent, les stratégies de réplication dans de tels environnements devraient également assurer une qualité du service du point de vue locataire (QoS) et une rentabilité économique du point de vue fournisseur. Dans ce contexte, nous présentons une étude comparative des stratégies existantes de réplication de données dans les systèmes cloud. Par ailleurs nous proposons une autre classification des stratégies de réplication dans les systèmes cloud basées sur les cinq critères suivants [40]:

- Nature de la réplication de données *(statique vs. dynamique)*([41];[42]).
- Équilibrage de charge de travail *(réactive vs. proactive)*([43];[44]).
- Orientation du bénéfice *(fournisseur vs. client)*([45];[46]).
- Fonction objective ([47];[48];[49]).

À la fin de ce chapitre, nous évoquerons et discuterons de quelques facteurs importants qui ont des influences sur la performance des stratégies de réplication de données dans les systèmes Cloud.

2.2. La Réplication de données dans les systèmes cloud

La plupart des stratégies de réplication de données proposées dans les systèmes cloud visent à augmenter la disponibilité des données, diminuer le temps d'accès aux données, réduire la latence d'accès et augmenter la fiabilité du système cloud.

Bien que le coût du stockage soit devenu moins important, certaines stratégies de réplication assument un stockage illimité ce qui irréaliste tandis que d'autres assument une quantité de stockage fixe et limitée. La réduction de la latence d'accès est également un objectif important des stratégies de réplication car elle réduit le temps d'exécution d'une tâche [50]. La plupart des stratégies de réplication nécessitent un compromis entre différents critères qui affectent les performances du système. Un grand nombre de stratégies de réplication de données ont été proposées dans la littérature. Dans ce qui suit elles sont classifiés selon différents attributs.

2.2.1. Classifications existantes

Un nombre significatif des stratégies de réplication de données ont été proposées dans la littérature qui ont été classées selon les caractéristiques suivantes :

2.2.1.1. Réplication statique vs. réplication dynamique

Lorsque le modèle de trafic réseau est stable et prévisible, la stratégie de réplication statique est la meilleure option. Le nombre et l'emplacement des répliques sont prédéterminés pour atteindre le niveau de performance souhaité. Les coûts associés à la stratégie de réplication statique sont directement proportionnels au nombre des répliques actives [51]. En revanche, avec le mécanisme de réplication dynamique, les répliques sont créées et supprimées dynamiquement par le composant système dédié en réponse aux modifications du mode d'accès utilisateur, de la capacité de stockage et de la consommation de la bande passante ([52];[53]).

2.2.1.2. Réplication centralisée vs. réplication décentralisée

Les mécanismes statiques et dynamiques peuvent être regroupés en algorithmes de réplication distribués ou centralisés. Les stratégies de réplication centralisée sont gérées par une autorité centrale d'une organisation pour planifier, surveiller et contrôler tous les aspects et processus de la réplication de données. De plus, tous les autres nœuds lui font rapport ([54];[55]). D'autre part, les stratégies de réplication décentralisées sont gérées et exécutées avec la collaboration et la coordination d'un certain nombre de nœuds ([56];[57];[35]).

2.2.1.3. Réplication déclenchée par le serveur vs. par Le client

Une autre façon de classification des stratégies de réplication est l'origine, client ou serveur, du déclenchement de l'événement de réplication ([38];[39]). Il peut être déclenché par un client (pull based replication) ou un serveur (push based replication).

2.2.1.4. Classification basée sur les fonctions objectives

Les auteurs dans [3] ont proposé une autre classification qui prend en compte la fonction objective de chaque stratégie de réplication. Ces objectifs comprennent :

- Les différentes formes de localité de données, telles que la localité temporelle, géographique et spatiale [56],
- Localité de niveau de réseau ([58];[35]),
- Modèle de coût [52],
- Comportement économique [59].

Les objectifs des stratégies de réplication sont souvent en conflit les uns avec les autres (minimisant les coûts par rapport à la réduction du temps d'accès) et / ou sans rapport (consommation d'énergie, temps de réponse). En raison de la nature de la fonction objective, la plupart des stratégies existantes ne satisfont pas simultanément plusieurs fonctions objectives.

2.2.1.5. Classification basée sur l'architecture du système

L'impact de l'architecture de grille de données sur les performances de réplication dynamique est étudié dans une étude de simulation. Les auteurs dans [31] ont classifié les stratégies de réplication en prenant l'architecture de grille de données cible comme un critère de classification. Ils ont conclu que des architectures plus souples donnent de meilleurs temps de réponse et offrent un accès aux données plus facile dans les applications où des demandes distantes fréquentes sont nécessaires.

2.2.2. La Classification proposée

Dans la littérature, les stratégies de réplication de données ont été classées en deux groupes principaux, les stratégies de réplication statique et dynamique [60]. En plus de cet aspect**,** nous proposons une nouvelle classification de stratégie de réplication de données dans des systèmes cloud selon les aspects suivants (voir figure 3.1) :

- Nature de la réplication de données *(statique vs. dynamique),*
- Équilibrage de charge de travail *(réactif vs. proactif),*
- Orientation du bénéfice *(fournisseur vs. client)*,
- Fonction objective.

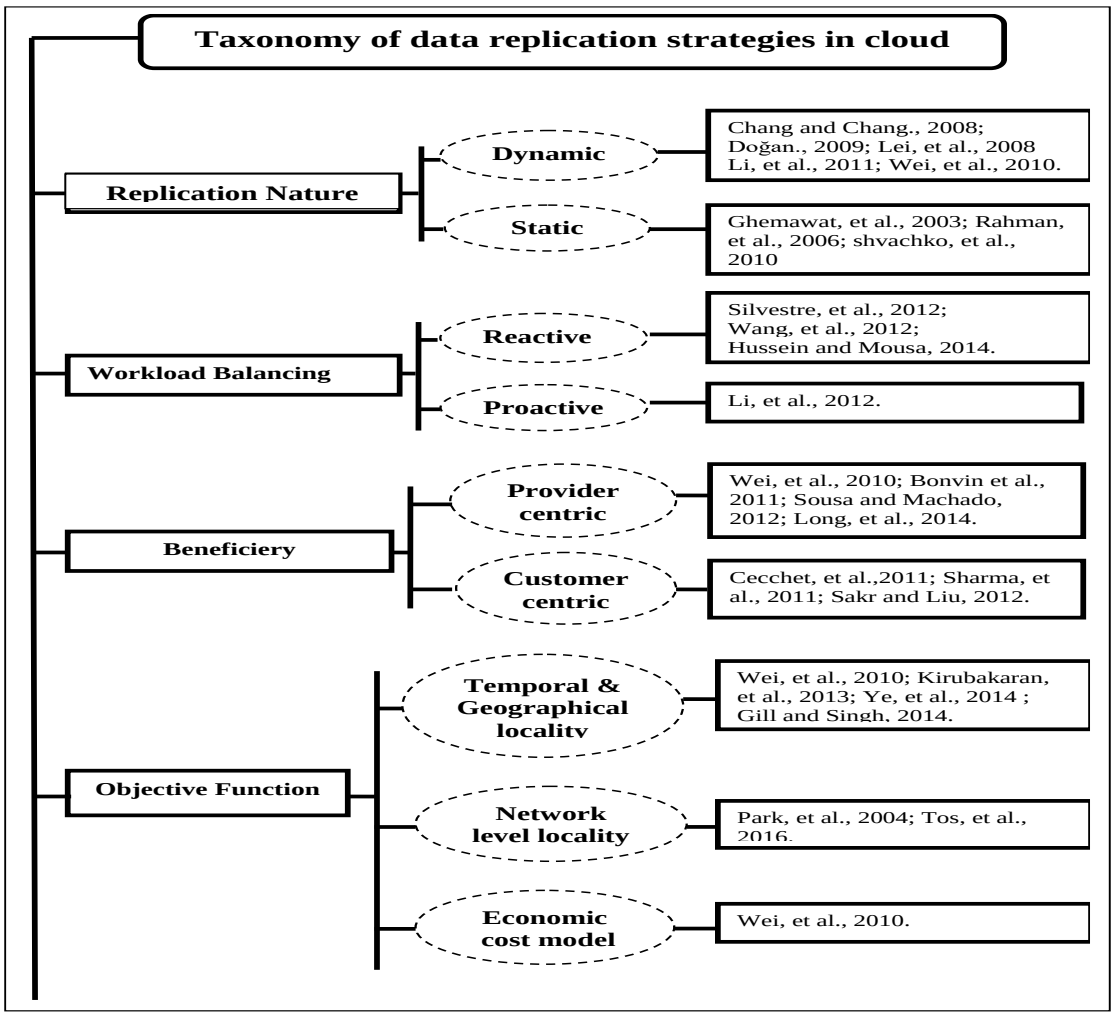


Figure 2.1. Taxonomie des stratégies de réplication de données dans les systèmes cloud.[40]

2.2.2.1. Nature de la réplication de données *(statique vs. dynamique)*

Toutes les stratégies de réplication de données peuvent être qualifiées de: statiques ou dynamiques. Ces stratégies sont formulées par trois facteurs importants: la stabilité du modèle d'accès de l'utilisateur, la capacité de stockage des nœuds de réseau et la consommation de la bande passante (le débit de transfert de données maximal d'un réseau). Lorsque le modèle de trafic réseau est stable et prévisible, la stratégie de réplication statique est la meilleure option. Avec les stratégies de réplication statique, le nombre et l'emplacement des répliques sont prédéterminés pour atteindre le niveau de performance souhaité défini par le SLA (accord de niveau de service), [43]. Le coût associé à la stratégie de réplication statique est directement proportionnel au nombre de répliques actives. [43]) ont proposé un algorithme statique de réplication de données dans les systèmes Cloud GFS (Google File System) qui offre une réponse rapide, une grande disponibilité et une grande efficacité, comme indiqué dans le tableau 2.1. Le système de fichiers Google est un système de fichiers distribués à un grand nombre de clients. Les avantages de GFS sont obtenus au détriment de plusieurs ressources, telles que l'augmentation du stockage et l'augmentation de la consommation d'énergie [61].

Replic strategy	GFS(Ghemawat et al 2003)	HDFS(Shvachko et al 2010)	RTRM(Bai et al 2013)	CDRM(Wei et al 2010)
	Static	Static	Dynamic	Dynamic
Advantages	- Low resp. time - High availability - High efficiency	- Low resp. time -High availability	- High performance - Low resp. time - Low data access - Low energy consump. - High availability	- High availability -High load balancing. -High perf -Low access cost - Low storage space
Disadvantages	- High replication cost - High energy consumption	- High replicationcost	- Low reliability - Low load balancing - High repl. cost	-Economic aspects neglected -Low reliability - High energy consumption -Low resp. time
Measured metrics				
Availability	Yes	Yes	Yes	Yes
Resp. time	Yes	Yes	Yes	No
Energy consump.	No	No	Yes	No
Repl. cost	No	No	No	No
Bandwidth consump.	No	No	Yes	No
Storage consump.	No	No	No	Yes
Important Factors				
Storage space	Limited	Limited	Limited	Limitted
Replica factor	Static	Static	Adjusted	Adjusted
Bandwidth consump.	Low	-	High	-
Balanced Workload	-	-	Low	High
Access Latency	-	-	Low	-

Tableau 2.1. Comparaison de certaines stratégies statiques et dynamiques.

En revanche, avec le mécanisme de réplication dynamique de données, les répliques sont créés et supprimées par le composant système dédié en réponse aux modifications du mode d'accès de l'utilisateur, de la capacité de stockage et de la consommation de la bande passante ([62];[63];[32]). Par conséquent, le système dédié prend des décisions intelligentes pour créer et supprimer des répliques en fonction de l'emplacement des données et le nombre optimal de répliques, et aussi en fonction des informations du modèle d'accès, de la capacité de stockage et de la consommation de la bande passante. Cependant, il présente certains inconvénients, tels que la difficulté de collecter des informations précises sur l'exécution de tous les nœuds hôtes dans une infrastructure cloud complexe. [42] ont proposé une méthode de réplication dynamique RTRM (Response Time based Replica Management). L'élément central de la stratégie RTRM est le temps de réponse du service d'accès au fichier unique, calculé par le nombre d'accès utilisateur simultanés. En utilisant un seuil pour le temps de réponse, RTRM prédit de manière dynamique la bande passante entre les serveurs de répliques et effectue la sélection de répliques en conséquence. Une autre méthode de réplication dynamique, la gestion rentable de la réplication dynamique (CDRM), offre une disponibilité rentable

tout en améliorant l'équilibrage de la charge et les performances du stockage dans le cloud [32]. Le tableau 2.2 compare quatre stratégies sélectionnées parmi des stratégies dynamiques et statiques. Les lecteurs sont invités [60] à consulter la liste complète des mécanismes de réplication statiques et dynamiques.

2.2.2.2. Équilibrage de la charge de travail *(réactif ou proactif)*

Puisque l'élasticité est l'une des principales caractéristiques associées à le cloud computing, l'équilibrage efficace et évolutif de la charge de travail est un élément essentiel de la gestion des ressources dans cet environnement [64]. Les algorithmes d'équilibrage de la charge de travail ont pour objectif de répartir les charges de travail sur un certain nombre de serveurs afin de minimiser le temps moyen nécessaire à l'exécution de ces tâches.

L'équilibrage de la charge de travail peut être proactif [63] ou réactif ([43];[65];[44]). Les techniques proactives sont basées sur la prédiction des nœuds alloués pour recevoir de nouvelles répliques. Un nouveau mécanisme rentable de gestion de la fiabilité des données dans le cloud, appelé PRCR (Contrôle dynamique de disponibilité des réplicas) est présenté pour maintenir la fiabilité des données [63]. La stratégie PRCR s'avère très efficace pour réduire la consommation d'espace de stockage et les coûts de stockage de manière significative dans le cloud. [47] ont proposé une approche économique pour devant permettre d'adapter dynamiquement les ressources du cloud de diverses applications. Leurs objectifs en termes de performance et de disponibilité des contrats de nveau de service sont fournis en présence de charges ou de d'échecs variables.

En revanche, les mécanismes réactifs détectent des situations de surcharge et y réagissent en définissant des seuils. AREN(schéma de réplication adaptative pour les réseaux de périphérie) est un schéma de réplication adaptative réactif pour le stockage en nuage [43]. Il a été conçu pour fournir un package ou contenu Web et ce sur demande expressive des utilisateurs finaux afin d'effectuer des mesures de qualité de service (QoS) strictes. Ils représentent la métrique de base des mesures de performance dans le réseau: disponibilité, perte, délais et utilisation [66]. La tâche visant à effectuer des mesures strictes de qualité de service n'est pas facile car la popularité du contenu varie avec le temps, et l'accord de niveau de service (SLA) inclut à la fois les paramètres de taux de transfert et de latence. AREN adapte le degré de réplication en fonction des contrats SLA stricts et de la croissance de la popularité du contenu.

AREN empêche ainsi, la plupart des violations de la SLA. HLES (lissage linéaire et exponentiel de Holt) est une autre stratégie de réplication adaptative réactive dans l'environnement en nuage. Elle améliore la fiabilité des fichiers de données en fonction de la prédiction de l'accès utilisateur aux blocs de chaque fichier. Il identifie le fichier populaire pour la réplication, basé sur l'analyse de l'historique d'accès récent aux données [44].

Une autre stratégie proactive de réplication de données dynamique, CAGW-DP(Closest Access Greatest Weight avec suppression proactive) utilise deux idées:

employant des enregistrements d'accès historiques qui sont utiles pour sélectionner les fichiers à répliquer, et une méthode de suppression proactive pour contrôler le nombre de répliques. La méthode de suppression atteint un équilibre optimal entre le temps d'accès réel et le temps système de mise à jour d'écriture [65].

RepliCstrategy	PRCR (Li et al 2012) Proactive	AREN (Silvestre et al 2012) Reactive	HLES (Hussein & Moussa 2014) Reactive	CAGW-DP(Wang, et al 2012) Reactive
Advantages	- Low replication cost - High reliability	- High performance - Low penalty cost	- High availability - High reliability - High load balance - Low response time - Low replication cost	- High load balance
Disadvantages	- High response time	- Economic aspects neglected.	-Low speed data access - High user waiting time	- Low speed data access
Measured metrics				
Availability	Yes	Yes	Yes	Yes
Response time	No	Yes	Yes	No
Energy consump.	No	No	No	No
Repl. cost	Yes	No	Yes	No
Bandwidth consump.	No	No	No	No
Storage consump.	Yes	Yes	No	Yes
Important Factors				
Storage space	Limited	Limited	Limited	Limited
Replica factor	-	-	-	Adjusted
Bandwidth consump.	Low	Low	Low	Low
Balanced Workload	High	High	High	High
Access Latency	-	-	Low	-

Tableau 2.2. Comparaison des stratégies basées sur l'équilibrage de la charge de travail

2.2.2.3. Orientation du bénéfice *(fournisseur vs. client)*

La plupart des stratégies de réplication proposées pour les systèmes cloud dans la littérature sont principalement axées sur la perspective des fournisseurs de cloud ([32];[47];[46];[67]) afin de minimiser la consommation de ressources tout en respectant les SLA avec les utilisateurs Cloud. La stratégie de gestion rentable de la réplication dynamique (CDRM) est un exemple de stratégie centrée sur le fournisseur, qui vise à déterminer le nombre minimal de réplicas pour chaque ensemble de données en fonction des exigences de [46].

Il existe également certaines stratégies de réplication centrées sur le client. [68] a proposé un système, Dolly, pour l'approvisionnement dynamique des répliques de bases de données sur les plates-formes cloud. Le système a été nommé en référence au

mouton Dolly, le premier mammifère à être cloné, et il adapte la politique ou stratégie d'approvisionnement aux spécificités de l'infrastructure du cloud et aux exigences de l'application. Il s'agit d'un outil d'approvisionnement dynamique permettant d'ajouter de manière autonome de la capacité dans des applications cloud à plusieurs niveaux, à mesure que la charge de travail augmente. [69] ont proposé Kingfisher, un système économique qui offre un soutien efficace pour l'élasticité dans le cloud, où chaque client optimise individuellement ses besoins en terme de capacité en choisissant la configuration de serveur qui correspond le mieux à ses besoins. Il a été prouvé que Kingfisher était capable de réduire le coût du serveur virtuel et de réduire considérablement le temps de transition vers une nouvelle configuration dans le cloud. [45] présentent un outil d'approvisionnement dynamique centré sur le SLA et axé sur le consommateur pour les bases de données cloud. Cette stratégie de réplication gère de manière flexible les performances et les exigences techniques des applications du client. elle facilite également l'approvisionnement adaptatif et dynamique du niveau de base de données des applications logicielles, en évitant le coût de toute violation des SLA et en contrôlant le coût monétaire des ressources informatiques allouées.

Replicstrategy	**RepliC(Sousa& Machado 2012)**	**SLA-based Provisioning (Sakr& Liu 2012)**	**Kingfisher (Sharma et al 2011)**	**Dolly (Cecchet et al 2011)**
	Provider-centric	**Customer-centric**	**Customer-centric**	**Customer-centric**
Advantages	**- Low penalty cost. - High load balance**	**-High load balance - Low penalty cost**	**-High elasticity. -Low replication cost - Low response time**	**-Low replication cost**
Disadvantages	**-Storage and migration costs neglected - Absence of replica placement strategy**	**- Economic aspects neglected**	**-Economic aspects neglected**	**- Unbalanced load**
Measured metrics				
Availability	**Yes**	**Yes**	**Yes**	**No**
Resp. time	**Yes**	**No**	**Yes**	**Yes**
Energy consump.	**No**	**No**	**No**	**No**
Repl. Cost	**No**	**No**	**Yes**	**Yes**
Bandwidth consump.	**No**	**No**	**No**	**No**
Storage consump.	**No**	**No**	**No**	**Yes**
Important Factors				
Storage space	**Limitted**	**Limited**	**Limited**	**Limited**
Replica factor	**Adjusted**	**-**	**-**	**Adjusted**
Bandwidth consump.	**Low**	**Low**	**Low**	**Low**
Balanced Workload	**High**	**High**	**High**	**Low**
Access Latency	**-**	**-**	**Low**	**Low**

Tableau 2.3. Comparaison des stratégies centrées sur le client et celles centrées sur le fournisseur.

2.2.2.4. Fonctions objectives

Les stratégies de réplication de données peuvent également être classées selon la fonction objective :

- Différentes formes de localisation de données en considérant la popularité des données ([32];[48];[70]),
- Une localité de réseau à large bande passante de réseau [49],
- Un modèle de coût qui choisit une réplication tout en minimisant son coût ([32];[47]).

a. Localité temporelle et géographique

La localité fait référence à la localité de référence traitant les problèmes d'accès aux données. La localité temporelle dans la réplication de données fait référence au concept selon lequel les données auxquelles on a accédé récemment seront à nouveau accessibles dans un intervalle de temps plus court avec une probabilité plus élevée. La localité spatiale (géographique) renvoie au concept selon lequel la probabilité d'accéder aux données est plus élevée si les données qui lui sont voisines viennent d'être consultées.

Une nouvelle stratégie de réplication dynamique des données, D2RS, est proposée par [32] pour augmenter la disponibilité du système et minimiser la consommation de bande passante du système cloud. Cette stratégie est basée sur le principe de la localité temporelle. Les données les plus populaires sont déterminées en analysant le modèle d'accès historique des données. Lorsque la popularité des données dépasse un seuil dynamique, l'opération de réplication sera déclenchée. En outre, le nombre de répliques sera déterminé en fonction de la disponibilité du système et de la probabilité de défaillance. Une nouvelle réplique sera créée sur les sites proches pour les utilisateurs qui génèrent le plus de demandes de données.

[48] ont proposé une stratégie modifiée de réplication de données dynamiques (MDDRS) pour déterminer un nombre et un emplacement raisonnables de répliques. Cette stratégie est une version améliorée et modifiée de la stratégie de réplication dynamique de données (DDRS). La mise à jour synchrone et asynchrone du fichier de données de réplique est adoptée lors de la mise à jour du centre de données principal. Ils ont comparé la stratégie de réplication de données dynamiques modifiée et la stratégie normale de réplication de données dynamiques [32] en termes de disponibilité du système. La comparaison des performances a montré que la disponibilité du système D2RS était supérieur à celle du MDDRS.

[70] ont proposé une stratégie de création de répliques dynamiques à deux couches et basée sur le géo-cloud à deux couches, nommée TGstag, afin de résoudre le problème de la modification du modèle d'accès des utilisateurs en réponse à la popularité des données. TGstag tente de minimiser la consommation de bande passante entre centres de données croisés et le temps d'accès moyen avec des contraintes de stratégie et de capacité de nœud de produits. cette stratégie réduit considérablement la bande passante

entre centres de données et le temps moyen d'accès en lecture. Ce travail est basé sur l'observation que la plupart des applications ont une localité temporelle et géographique. En prenant en compte ces deux localités , elles ont utilisé les statistiques d'accès passées pour prédire les profils d'accès des utilisateurs futurs.

[71] ont proposé un algorithme, stratégie de réplication et de rééquilibrage dynamique en fonction des coûts (DCR2S) pour optimiser le coût de la réplication en utilisant le concept du problème du « sac à dos ». Le système est conçu pour comprendre la relation entre un certain nombre de répliques, le coût de la réplication et la disponibilité. Ensuite, un algorithme détermine quel fichier doit être répliqué et quand il doit être répliqué en explorant la localité temporelle. Les résultats expérimentaux ont prouvé que DCR2S peut optimiser le coût de la réplication. Il a atteint un taux effectif élevé d'octets système, et il est assez efficace dans une architecture du système cloud hétérogène.

b. Localité au niveau du réseau

Une nouvelle stratégie de réplication dynamique BHR, est proposée pour réduire le temps d'accès aux données en évitant les congestions du réseau dans un réseau de grille de données [58]. La stratégie BHR tire parti du concept de «localisation au niveau du réseau», où le fichier requis est situé sur le site qui dispose d' une large bande passante vers le site d'exécution du travail. La stratégie BHR utilise d'autres techniques d'optimisation en termes de temps d'accès aux données. BHR étend l'étude d'optimisation des répliques au niveau du site actuel au niveau du réseau.

Une autre étude de [49] exploite la localité de la bande passante tout en proposant une stratégie de réplication de données qui vise à satisfaire à la fois la performance des locataires et le profit des fournisseurs. Le temps d'accès aux données et la consommation de la bande passante réseau (NB) peuvent être minimisés en créant des répliques principalement au niveau des nœuds ayant le NB le plus large vers le nœud qui soumet une requête. Une opération de réplication de données est déclenchée uniquement si le temps de réponse de requête estimé dépasse un seuil de temps de réponse convenu et que cette réplication est alors rentable pour le fournisseur. D'autre part, lorsque le temps de réponse est satisfait au fil du temps, une réplique qui n'est pas utilisée est retirée du cloud.

c. Modèle de coût économique

La stratégie du CDRM (gestion économique de la réplication dynamique) propose de fournir des données rentables. Elle permet également d'améliorer les performances et l'équilibrage de la charge du stockage dans le cloud en ajustant le nombre de répliques et leur emplacement dans le stockage de données en fonction du changement de la charge et de la capacité des nœuds [32]. Son objectif principal est d'améliorer les performances et l'équilibrage de la charge de stockage dans le cloud. Le CDRM peut redistribuer dynamiquement les charges de travail entre les nœuds de données dans le cloud. Une réplique, dans la stratégie CDRM, est placée sur le nœud avec la plus faible probabilité de blocage des nœuds de données afin de réduire le temps d'accès aux données.

RepliCstrategy	CDRM (Wei et al. 2010)	Bonvin et al. 2011	DCR2S (Gill & Singh 2015)	MDDRS (Kiruba et al. 2013)
Advantages	- High availability -High load balancing -High perf -Low access cost - Low storage space	-High availability -Low repl. cost	- High availability - Low repl cost - High reliability	- High availability - Low bandwidth consumption
Disadvantages	-Low reliability - High energy consumption -Low resp time -Economic aspects neglected	- Low load balancing - High resp. time -Geographical diversity of replica location	- Low load balancing - High resp. time	- Economic aspects neglected
Measured metrics				
Availability	Yes	Yes	Yes	Yes
Resp. time	No	No	No	No
Energy consump.	No	No	No	No
Repl. Cost	No	Yes	Yes	No
Bandwidth consump.	No	No	No	Yes
Storage consump.	Yes	No	No	No
Important Factors				
Storage space	Limited	Limited	Limited	Limited
Replica factor	Adjusted	-	Optimal Nbr	-
BandwidthConsump	-	-	-	Yes
Balanced workload	High	Low	Low	-
Access latency	-	-	-	-

Tableau 2.4. Comparaison des stratégies basée sur la fonction objective.

2.3. Discussion

La plupart des stratégies de réplication statiques génèrent une consommation de stockage, un coût de réplication, une consommation d'énergie et une consommation de bande passante élevée car le nombre de répliques est fixé au départ par l'utilisateur. En raison de la nature dynamique des systèmes cloud, les stratégies de réplication dynamique sont plus adaptées dans les systèmes cloud.

En ce qui concerne le traitement de l'observation de la charge de travail, il n'y a pas de consensus dans la littérature sur la stratégie à adopter, c'est-à-dire, réactive ou proactive. Par conséquent, certaines solutions, par exemple le système Kairos [72], utilisaient une solution hybride via la surveillance et la consolidation de la base de données en fonction de la charge de travail.

La plupart des stratégies proposées se concentrent sur un le fournisseur de services cloud qui vise à maximiser l'utilisation des ressources et des bénéfices des fournisseurs. Seules quelques œuvres sont centrées sur le client, ce qui considère les coûts du point de vue du locataire sans négliger pour autant les profits du fournisseur. [73] modélisent

la situation comme un problème d'optimisation à objectifs multiples, et le coût dépensé par un locataire est l'un de ces objectifs.

Déterminer le nombre requis de répliques constitue un autre aspect de la classification. Malgré une différence entre les stratégies visant à déterminer un facteur de réplique optimale et celles qui ajustent le facteur de réplique, les deux stratégies sont basées sur la charge de travail. En outre, le coût de la réplication augmente dans les deux catégories. De plus, la bande passante et la consommation d'énergie sont élevées en cas d'ajustement du facteur de réplique, car il y a interaction continue et des progrès dans les travaux visant à réaliser un équilibre de charge élevé et une haute performance.

Nous avons également classé les stratégies de réplication selon leurs fonctions objectives, en fonction du rôle de ces stratégies lors de la construction d'un système de gestion des répliques. La plupart des stratégies traitent la localité des données temporelles. D'autres stratégies traitent de l'utilisation d'un modèle de coût.

Seules quelques stratégies traitent d'un modèle de coûts économiques qui garantit au fournisseur certains avantages. À notre connaissance, la plupart des stratégies de réplication proposées dans les systèmes cloud ne répondent qu'à une fonction objective spécifique et ne peuvent pas les utiliser toutes Cependant, une fois regroupées, ces fonctions d'objectif répondent mieux aux problèmes de réplication de données, en particulier lorsqu'elles satisfont à la fois aux exigences des locataires, c'est-à-dire à la garantie QoS, et au profit du fournisseur. En revanche, la plupart des stratégies de réplication incluses dans cette étude visent à obtenir de meilleures performances en gérant le placement et le nombre optimal de répliques tout en négligeant les aspects économiques et principalement la gestion des pénalités.

2.4. Introduction aux simulateurs

Le tableau 2.5 décrit les avantages, les inconvénients et certaines autres caractéristiques des simulateurs examinés. Lorsque l'on compare ces simulateurs, tous, excepté green cloud, sont implémentés avec Java. Par conséquent, ils peuvent être exécutés sur n'importe quelle JVM installée sur une machine, tandis que green cloud se combine entre C++ et OTcl, et IcanCloud est en C ++. CloudSim est le simulateur le plus sophistiqué et le plus utilisé avec les fonctionnalités de bibliothèque cloud les plus riches. Nous prévoyons que CloudSim continuera d'être le principal outil de simulation pour les recherches futures sur les problèmes d'informatique en cloud. D'autre part, la faiblesse critique de CloudAnalyst est l'incapacité de fournir des informations concernant la consommation d'énergie du centre de données. CloudReport fournit des informations sur l'utilisation de la puissance du centre de données, mais il ne peut pas fournir la consommation d'énergie de chaque composant. Enfin, tous les simulateurs du tableau 3.5 sont open sources et donc disponibles en téléchargement public.

Cloud simulators	CloudSim	CloudAnalyst	GreenCloud	CloudReport	IcanCloud
Advantages	-Rich of cloud libraries functionalities. -It has many extensions.	-Based on GUI. -Provide outputs in graphical form	-Models detailed energy consumptions for different physical components	-Based on GUI. -Provide outputs in graphical form -Allow parallel execution over several machines	-Provide outputs in graphical form
Disadvantages	-Absence of a GUI. -Very complex	-Unable to provide information of power consumption of DC	-Absence of a GUI. -Large memory space	-The power consumption by each component is not provided.	-Absence of a GUI
Characteristics					
Platform	SimJava	Extended SimJava	NS2	Extended SimJava	OMNET++
Output form	Log	Table + pie chart	Log+pie chart	Log+pie chart	Pie chart
Language	Java	Java	C++/OTcl	Java	C++
Availability	Open source	Open source	Open source	Open source	Open source
Memory space	Small	Small	Large	Small	Medium

Tableau 2.5. Comparaison des simulateurs de cloud.

2.5. Facteurs de performance des stratégies de réplication de données

Pour atteindre les performances tout en gérant un processus de réplication de données, nous devons toujours assurer que le bénéfice d'une stratégie donnée est plus élevé que le coût de la réplication [74].

Adopter une stratégie plutôt qu'une autre dépend de plusieurs facteurs à privilégier afin d'obtenir une performance optimale. Par conséquent, les compromis entre les facteurs comme la latence d'accès, l'état du réseau (par exemple, la bande passante) et le coût de stockage dans les nœuds doivent être pris en compte. En conséquence, le coût de chaque stratégie de réplication dépend de la décision de favoriser un facteur par rapport aux autres. Dans ce qui suit, nous énumérons les facteurs les plus importants qui influent sur la performance de toute stratégie de réplication.

2.5.1. Granularité optimale de la réplication de données

La détermination de la granularité appropriée des données à répliquer s'avère cruciale lors de la réplication de données avec un objectif de performance.[74] démontre que la granularité optimale dépend des applications. Pour les nœuds qui stockent des pages Web statiques par exemple, la prise en charge d'une stratégie de réplication par page entraîne une meilleure évolutivité et de meilleures performances. [74] conclut que la réplication pour la performance nécessite de différencier les stratégies de réplication pour les unités de données plus petites.

2.5.2. Latence d'accès

La réduction de la latence d'accès constitue un facteur important pour réduire le temps d'exécution du travail. Ceci est obtenu en partageant des informations entre tous les nœuds afin de savoir quelles données doivent être répliquées et où placer la nouvelle réplique [55]. [75] ont réduit également la latence d'accès en sélectionnant la meilleure réplique lorsque plusieurs nœuds contiennent des répliques. L'algorithme proposé est basé sur le temps de réponse qui peut être déterminé en considérant le temps de transfert de données, la latence d'accès au stockage et la distance entre les nœuds.

2.5.3. Consommation de la bande passante

Certaines stratégies de réplication ne considèrent pas une consommation de bande passante optimale puisque le but principal est d'améliorer la disponibilité des données. Cependant, ce facteur est très important pour assurer la performance. En fait, un transfert fréquent de données peut entraîner des contraintes sur les ressources du réseau, ce qui peut avoir un impact sur les performances du système. Cela motive la proposition de réplication stratégies basées sur le niveau du réseau.

2.5.4. Equilibrage de charge de travail

Le placement de répliques dans des emplacements optimaux permet d'optimiser la charge de travail du système et de minimiser le temps d'exécution du travail. [76] ont proposé une réplication dynamique basée sur p-median qui trouve un placement p réplica qui minimise la distance entre le nœud demandeur et les nœuds contenant des répliques. [77] se concentre sur le placement optimal des répliques de sorte que la charge de travail des répliques soit équilibrée pour l'architecture multi-niveaux.

2.5.5. Capacité de stockage

Bien que les coûts de stockage soient devenus faibles dernièrement, les stratégies de réplication doivent supposer une quantité de stockage fixe pour garantir le réalisme. Les performances de réplication dépendent de manière significative de la taille du stockage disponible sur différents nœuds et de la bande passante entre ces nœuds. En conséquence, il existe un compromis entre la disponibilité du stockage et la disponibilité de la bande passante du réseau [36]. Une solution consiste à considérer des algorithmes de remplacement de réplique bien conçus [78].

2.5.6. Nombre optimal de répliques

La définition d'un nombre optimal de répliques, afin d'éviter la réplication inutile, est un paramètre important lorsque les stratégies de réplication atteignent des performances. En fait, le maintien d'un nombre accru de répliques peut générer un surcoût dans le système. [79] se concentre sur le placement optimal des répliques pour l'architecture hybride. Il essaie de maintenir une charge de travail équilibrée sur tous les nœuds en proposant un algorithme pour trouver le nombre optimal de répliques. Ensuite, un autre algorithme place les répliques dans des emplacements optimaux si le nombre de

répliques et la charge de travail maximale autorisée pour chaque réplique a été déterminé.

La plupart des stratégies de réplication dans la littérature considèrent que l'amélioration de la disponibilité et la réduction du temps d'exécution du travail constituent l'objectif principal de ces stratégies. Bien qu'il existe un compromis entre certains facteurs, tous ces facteurs doivent être pris en compte simultanément. Conserver les données à proximité de l'utilisateur, c'est-à-dire réduire les coûts d'accès, ne devrait pas se faire au détriment de la congestion du réseau. De nombreux travaux ont également conclu qu'une bonne stratégie de réplication doit être basée sur un algorithme de placement de réplique efficace avec un nombre optimal de répliques alors que le choix des nœuds contenant ces répliques ne doit pas se faire au détriment de la charge du système.

2.6. La réplication de données dans les systèmes de base des données NoSQL orientées documents : Mongodb

2.6.1. Introduction à Mongodb

Mongodb est un système de base de données NoSQL open source orienté document développé par 10gen en 2007 et écrit en langage de programmation C ++. Il est distribuée sous licence AGPL (licence libre) et est très adapté aux applications web.

Mongodb a été adoptée par plusieurs grands noms de l'informatique, tels que Foursquare, SAP, ou bien même GitHub. Il gère des collections de documents BSON, fournit des performances élevées. Mongodb réplique les données sur plusieurs serveurs avec le principe de maître-esclave, permettant ainsi une plus grande tolérance aux pannes, une haute disponibilité, une évolutivité facile et l'auto-fragmentation.

Le concept principal dans Mongodb est le document qui présente l'unité principale des données. Cela équivaut à un enregistrement dans un système de gestion de bases de données relationnelle (RDBMS). Une collection peut être équivalente à une relation [94]. Le tableau suivant montre la relation de la terminologie RDBMS avec Mongodb :

RDBMS	**Mongodb**
Base de données	Base de données
Table	Collection
Tuple/Row	Document
Colonne	Champ
Table Join	Documents intégrés
Clé primaire	Clé primaire (id_clé par défaut fournie par Mongodb lui même)

Tableau 3.1. Equivalence entre les concepts dans RDBMS et Mongodb.

2.6.2. Les travaux connexes

Il existe plusieurs travaux qui se concentrent sur la base de données nosql Mongodb. Dans cette section, nous présenterons un aperçu de quelques travaux récents dans ce sujet.

Les données sont réparties d'une manière inégale entre les fragments en utilisant l'algorithme existant de balance dans Mongodb. Afin de résoudre ce problème [87] ont proposé un algorithme appelé FODO (fréquence de traitement des opérations) qui prend en compte la charge des serveurs et la fréquence des opérations de données. L'algorithme proposé peut équilibrer les données entre les fragments. Ce qui permettra d' améliore les performances du cluster (lecture et écriture).

Dans un autre travail les auteurs [88] ont analysé les différents mécanismes de réplication utilisés dans Mongodb. Ils ont fourni une explication approfondie des deux modèles de réplication utilisés: (i) le modèle Maitre/esclave, et (ii) le modèle des jeux de répliques.

[90] ont mené une étude détaillée pour évaluer l'impact de la réplication de données biologiques sur deux différentes types de base de données NoSQL: orienté document *Mongodb* et orienté colonne *Cassandra*. Les résultats ont montré une amélioration de la disponibilité des données, mais une perte de performance du système. Certains paramètres dans le cluster et les facteurs de réplication peuvent certainement affecter les résultats. L'impact du facteur de réplication sur le temps d'exécution était plus élevé.

Dans [91] les auteurs évaluent et examinent les performances lorsqu'ils utilisent ou non la réplication sous différents niveaux de charge de travail et de cohérence avec des bases de données NoSQL MongoDB et Cassandra. En utilisant les résultats de benchmarking, ils affirment que la réplication et la cohérence ont un impact direct sur les performances du système. Par ailleurs [93] ont présenté une étude complète des différentes bases de données NoSQL existantes y compris Mongodb. Basée sur le modèle de données, l'interrogation et le modèle de réplication, cette recherche fournit des connaissances permettant aux utilisateurs de choisir la base de données NoSQL appropriée selon leurs besoins. En analysant les différentes propositions existantes qui traitent des mécanismes de réplication dans Mongodb, nous remarquons que la plupart des stratégies se concentrent sur l'évaluation de réplication et leur impact et en même temps la comparaison des différents modèles de la réplication. D'autres recherches ont permis de comparer les performances de Mongodb avec d'autres systèmes NoSQL populaires, ainsi que des systèmes de gestion de bases de données relationnelles. Cependant, aucun d'eux n'attire l'attention sur la gestion des répliques dans Mongodb lorsqu'elle est utilisée comme serveur principal par un fournisseur de services cloud. De plus, ils négligent le profit du fournisseur. Dans notre stratégie proposée, nous avons traité la gestion des répliques dans Mongodb afin de garantir à la fois la performance du système et la rentabilité du fournisseur de cloud.

2.6.3. Les stratégies de réplication dans Mongodb

Dans Mongodb il existe deux modes de réplication de données: Maître/Esclave et replicaset. Le mode de réplication a une forte influence sur le comportement du système, Il est nécessaire de comprendre le mode de réplication dans chaque modèle. Dans les deux modèles la réplication est asynchrone où une instance stocke les données mises à jour et transmet les données modifiées aux répliques de manière différée. [95]

2.6.3.1. Modèle Maître/Esclave

Dans le modèle maître / esclave, les opérations d'écriture sont effectuées uniquement sur le serveur maître, puis les informations sont transmises de manière asynchrone à l'ensemble des serveurs esclaves. (figure 3.1)

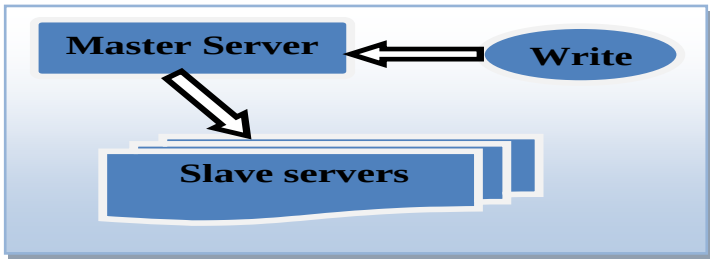


Figure 3.1. Modèle Maître/Esclave

Si un serveur esclave tombe en panne, il est possible de redémarrer l'instance. Les données seront synchronisées avec le serveur maître via un fichier journal afin d'éviter la congestion du réseau avec une migration des données depuis le début (figure 3.2).

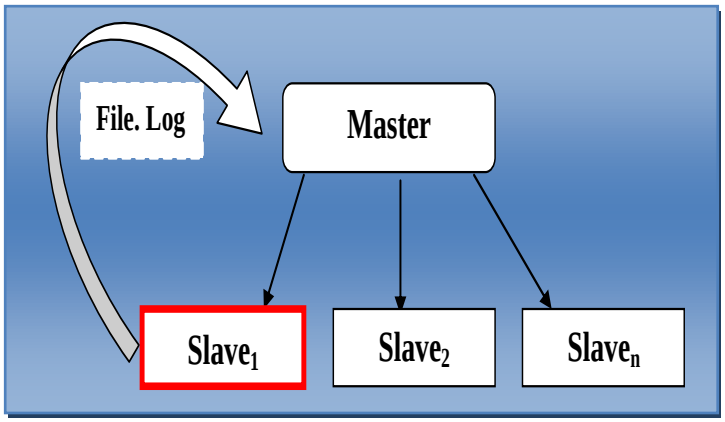


Figure 3.2. Cas de défaillance de l'esclave.

Dans le cas où le serveur maître tombe en panne, le système passe en mode lecture seule jusqu'à ce que le serveur maître redémarre, ce modèle de réplication permet un service de basculement en cas de panne. (figure 3.3).

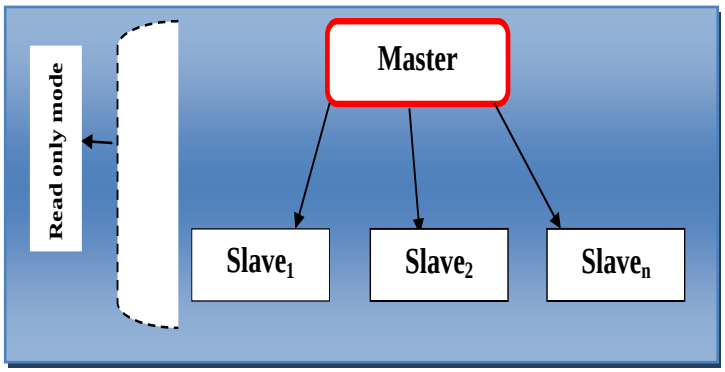


Figure 3.3. Cas de défaillance du serveur maître.

2.6.3.2. Modèle replicaSets

Cette approche considère tous les serveurs du même cluster en tant que serveur principal ou secondaire. Le modèle de jeu de répliques présente les améliorations de version du premier modèle (maître / esclave), il offre un équilibre de charge automatique au système.

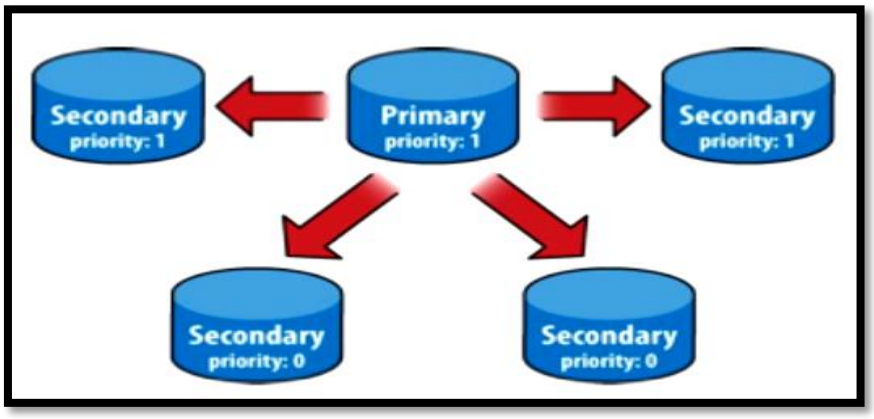


Figure 3.4. Modèle Replicasets. [95]

À tout moment, il n'y a qu'un seul nœud principal qui est essentiellement le maître, et le reste est secondaire. Nous distinguons trois types de nœuds existant dans Replicaset [95] :

- **Actif**
  - Stocker une copie complète des données répliquées.
  - Participer au vote pour l'élection du nouveau nœud principal (maître).
  - Peut devenir le nœud principal de l'ensemble.
- **Passif**
  - Stocker une copie complète des données.
  - Participer au vote.
  - Ne deviendra jamais le nœud principal de l'ensemble.

- **Arbitre**
  - Il ne stocke aucune copie des données répliquées.
  - ne peut pas devenir un nœud primaire.
  - Participer uniquement au vote.

En cas de défaillance du serveur principal, un nouveau serveur principal (maître) est élu par le même cluster, en fonction des formations mises à jour les plus récentes. Ce modèle de réplication a la capacité de surmonter la défaillance d'un nœud principal grâce à la planification automatique (basculement automatique) et à l'équilibrage automatique de la charge.

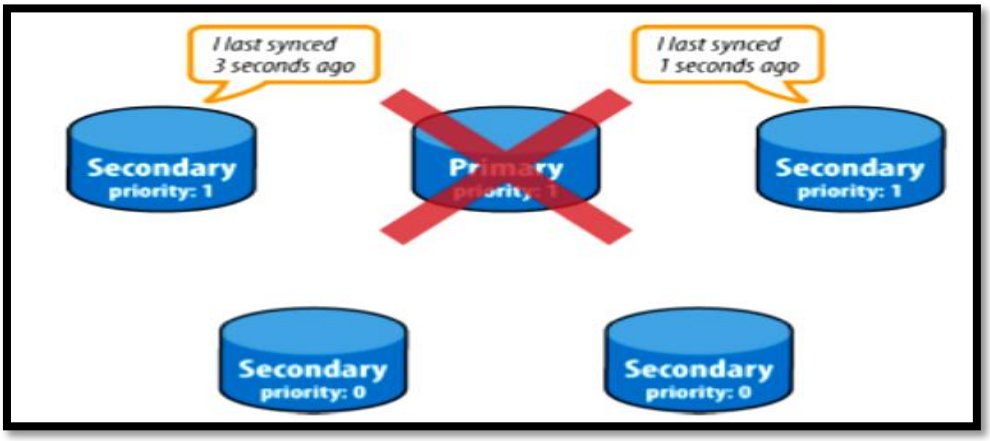


Figure 3.5. Cas de défaillance du serveur maître [95]

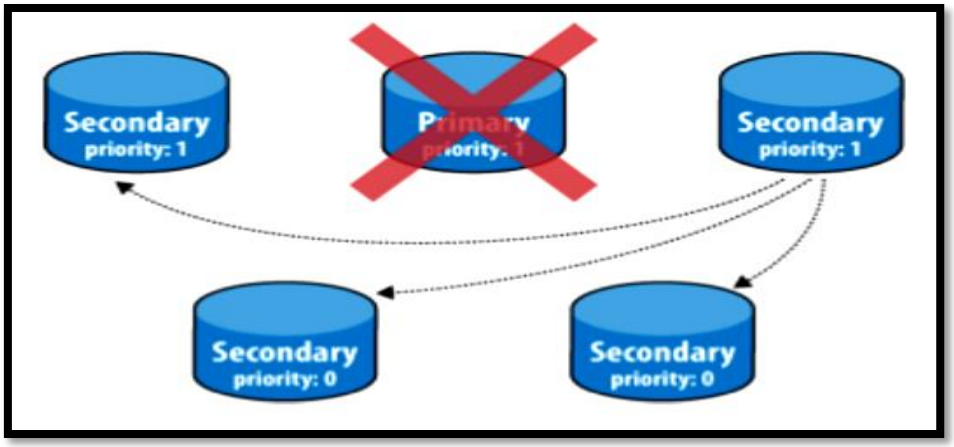


Figure 3.6. Election d'un nouveau nœud primaire [95]

La principale différence entre les deux modèles de réplication est qu'un ensemble de répliques peut basculer automatiquement lorsque le nœud principal est indisponible en sélectionnant un nouveau nœud principal à partir des nœuds secondaires existants en se basant sur le vote des nœuds situés dans le même cluster, en revanche avec le modèle maître-esclave toutes les répliques sont en lecture seule et mises à jour uniquement à partir du nœud principal après les modifications, ce qui peut causer des problèmes,

lorsque le nœud maître est en panne, dans ce cas il n'y a aucune possibilité d'écrire de nouvelles données. Ainsi, le modèle de réplication le plus approprié est le modèle de jeu de réplicas..

2.7. Conclusion

Dans ce chapitre, nous avons présenté une nouvelle taxonomie des stratégies de réplication de données dans les systèmes cloud (Figure 2). Nous avons classé les stratégies de réplication en fonction des quatre critères suivants:

- Modèle d'accès utilisateur*(statique vs. dynamique)*,
- Équilibrage de la charge de travail*(réactif vs. proactif)*,
- Bénéficiaire*(fournisseur vs. client)*,
- Fonction objective .

Les stratégies de réplication de données dans les systèmes cloud peuvent être évaluées et validées en fonction des objectifs tracés préalablement tels que la disponibilité, les performances et la tolérance aux fautes. Plusieurs simulateurs d'Cloud Computing ont été spécialement développés pour l'analyse des performances en cloud. Idéalement, une bonne stratégie de réplication doit tenir compte des éléments suivants:

- La réduction du temps d'accès
- La réduction de la consommation de bande passante
- La disponibilité des ressources de stockage
- Une charge de travail équilibrée entre les répliques
- Un algorithme de placement stratégique incluant un nombre ajusté de répliques

De plus, seules quelques-unes de ces stratégies mettent l'accent sur l'aspect économique, comme l'avantage du fournisseur et la gestion des pénalités. Les validations de la plupart de ces stratégies sont effectuées en comparant les résultats des stratégies proposées précédemment à l'aide de simulateurs existants. Par conséquent, la sélection d'une stratégie de réplication de données est un exemple classique d'un problème de prise de décision à critères multiples. Récemment, les auteurs dans [31] ont discuté ces critères dans les stratégies de réplication.

Nous pensons qu'un autre problème ouvert dans les stratégies de réplication dynamique permet d'atteindre les objectifs de performance tout en conservant un certain degré de réplication, c'est-à-dire un certain nombre de répliques. L'ajustement dynamique du nombre optimal de répliques prenant en compte les considérations économiques offre de nouveaux défis de recherche. Trouver le nombre optimal de répliques met également l'accent sur le placement des répliques. Étudier la défectivité inefficace du nombre optimal de répliques et les placer stratégiquement est toujours un problème en suspens et joue un rôle clé dans la réalisation de l'optimalité pour les consommateurs et les fournisseurs de ressources.

Au cours des dernières décennies , les chercheurs en systèmes d'informations ont constamment lutté pour gérer les problèmes de décision à critères multiples ([80];[81];[82]). La taxonomie que nous avons présenté peut-être un guide utile pour les responsables informatiques afin de choisir la stratégie de réplication de données pour leur organisation. Comme indiqué précédemment [60], nous ne pensons pas qu'il existe une stratégie optimale de réplication de données qui traite tous les problèmes liés à la réplication de ces dernières. De plus, nous pensons qu'il est hautement improbable de trouver une telle stratégie de réplication magique qui optimise un large éventail d'objectifs des acteurs (fournisseurs et clients) et des politiques organisationnelles. Au contraire, la stratégie de réplication pour toute organisation sera une décision qui satisfait plusieurs objectifs des entités impliquées. Il pourrait être très prometteur pour les futurs chercheurs d'aborder les problèmes des stratégies de réplication sous deux angles différents: la prise de décision à critères multiples et la gestion des ressources dans le cloud.

La gestion des ressources dans le cloud consiste à «allouer des ressources informatiques, de stockage, de réseau et (indirectement) énergétiques à un ensemble d'applications, de manière à atteindre conjointement les objectifs de performance des applications, de l'infrastructure des fournisseurs et des utilisateurs des ressources du cloud» [83]. Ces objectifs comprennent l'utilisation efficace des ressources dans les limites des accords de niveau de service, des performances des applications, de leur disponibilité, ainsi que du passage à l'échelle rentable des ressources disponibles. Plusieurs contraintes doivent également être prises en compte, notamment en matière de sécurité ou la conformité réglementaire. Dans ce contexte, la réplication de données est une technique utilisée pour la gestion des ressources dans le cloud. En conséquence, la réplication de données doit être formulée avec plusieurs objectifs de gestion ainsi que des objectifs inévitablement contradictoires des utilisateurs finaux et des fournisseurs.

Dans le chapitre qui suit, nous aborderons la contribution principale de cette thèse. Ainsi, lors de la description de la stratégie de réplication de données proposées, nous aborderons différents problématiques correspondantes aux questions suivantes : quelles données doivent être répliquées ?, quand déclencher l'événement de réplication ?, combien de répliques sont créés lors de chaque réplication ? et où placer les répliques nouvellement créées ? La stratégie proposée se base sur un modèle de coût et un modèle économique que nous discuterons dans le chapitre suivant.

CHAPITRE 3 : LA STRATEGIE DE REPLICATION DE DONNEES PROPOSEE

Sommaire

CHAPITRE 3 : LA STRATEGIE DE REPLICATION DE DONNEES PROPOSEE 59
3.1. Introduction 60
3.4. La topologie considérée 62
3.5. La stratégie proposée de la réplication de données 63
3.5.1. La décision de la réplication 64
3.5.1.1. Modèle de coût pour le traitement des requêtes 65
- Estimation du temps de réponse d'une requête 65
a. Estimation du temps CPU 66
b. Estimation du temps I/Os 66
c. Estimation du temps de communication 67
3.5.1.2. Modèle économique pour le traitement de requêtes 67
a. Estimation des revenus 67
b. Estimation des dépenses 68
3.5.2. La sélection des données à répliquer 68
3.5.3. Le placement des nouvelle répliques 69
3.5.4. La suppression de répliques 71
3.6. Conclusion 72

3.1. Introduction

La gestion des données volumineuses et hétérogènes est l'un des domaines de recherche les plus importants dans le cloud. Par conséquent les entreprises se sont tournées vers les environnements cloud pour héberger leurs applications et bases de données. Elles s'attendent à ce que les fournisseurs de cloud respectent un certain nombre d'objectifs de niveau de service (SLO) définis. Les plateformes et les logiciels sont offerts en tant que services avec des paramètres de paiement basés sur le principe du paiement à l'usage " modèle pay-as-you-go " [84]. Dans un tels modèle, l'utilisateur ne paye que ce qu'il consomme.

Les SLOs représentent les objectifs entre le fournisseur et ses clients, définis dans un accord de niveau de service (Service Level Agreement, SLA) [85]. Ce dernier présente un contrat juridiquement contraignant entre le fournisseur de cloud et son locataire. Le SLA spécifie quel service doit être fourni, comment il est maintenu, les temps, les lieux, les coûts, les objectifs SLOs (exemple : les performances) et les responsabilités des parties impliquées. Les SLOs sont des objectifs spécifiques et mesurables des SLAs, comme la disponibilité, la fréquence, le temps de réponse ou encore la qualité. Dans ce contexte, la performance constitue un SLO important. Le fournisseur paie une pénalité au locataire en cas de violation d'un SLA.

La réplication de données est une technique permettant la création des copies identiques de données (répliques) sur plusieurs serveurs. Elle permet d'améliorer la disponibilité des données, le temps de réponse et la tolérance aux pannes et réduit le trafic réseau [40]. Elle est fréquemment utilisée dans: (i) les systèmes de gestion de bases des données [26], (ii) les systèmes parallèles et distribués ([27];[28]), (iii) les systèmes à grande échelle, y compris P2P [29] et dans les grilles des données([30];[31]). Cependant, les stratégies proposées dans les systèmes traditionnels ex : les grilles des données ne sont pas adoptées dans les systèmes de cloud. Ils visent à obtenir les meilleures performances sans prendre en compte le profit des fournisseurs de cloud, qui cherchent à maximiser leurs profits. Ce qui motive la proposition de plusieurs stratégies de réplication dans les systèmes du cloud ([32];[45];[50];[49]). Les systèmes de gestion de bases de données relationnelles font face à de nombreux obstacles pour atteindre ces besoins. Par conséquent, l'utilisation de bases de données NoSQL (not-only SQL) devient nécessaire, en particulier lorsqu'il s'agit de données volumineuses et hétérogènes.

Les bases de données NoSQL ont pris une grande importance au cours des dernières années. Etant donné que les systèmes d'information génèrent d'énormes quantités de données, il est évident que, pour traiter de tels volumes de données, une capacité énorme est nécessaire en termes de ressources de stockage et de calcul. Alors que la croissance de la capacité est limitée par l'évolution du matériel et des technologies, la croissance du volume de données est en fait illimité. L'un des systèmes NoSQL les plus connu est Mongodb [86]. Afin d'améliorer les performances d'un tel système, certains

travaux ont focalisé leur attention sur l'amélioration des algorithmes d'auto-sharding, d'équilibre de charge et de passage à l'échelle automatique, par exemple ([87];[88];[89]). D'autres travaux analysent les différences existantes entre MongoDB et les systèmes de gestion de bases de données relationnelles, par exemple ([90];[30]). En outre, beaucoup de ces travaux ont été concentrés sur la performance en utilisant une analyse comparative tout en comparant les capacités de Mongodb avec d'autres systèmes NoSQL [91]. Les résultats ont montré que Mongodb offre de meilleures performances.

Cependant, la plupart de ces travaux se concentrent uniquement sur l'amélioration des performances du système. Par exemple, toutes les répliques sont en lecture seule et sont mises à jour uniquement à partir du nœud maître du modèle de réplication maître-esclave. Cela peut provoquer des problèmes lorsqu'un nœud maître tombe en panne. Pour surmonter ce problème et garantir les performances du système, le modèle ReplicaSet a été proposé. Il surmonte l'échec d'un nœud primaire par une technique dite '*auto scheduling'*. Ensuite, il offre un équilibrage automatique de la charge du système qui offre de meilleures performances [92]. Cependant, les coûts de réplication, les violations de SLA et les problèmes de bénéfice des fournisseurs ont été négligés.

Dans ce chapitre, nous présentons une nouvelle stratégie de réplication de données pour les systèmes NoSQL orientés documents *Mongodb*. Il vise à assurer à la fois la performance du système pour l'utilisateur et une rentabilité pour le fournisseur du cloud.

Une réplication d'un document dans une collection est considérée seulement si deux conditions sont satisfaites: (i) le temps de réponse estimé, juste avant d'exécuter une requête, est supérieur à un seuil de temps de réponse *RespT* inclus dans le contrat SLA puis, (ii) si cette réplication se produit, le fournisseur devrait avoir un profit économique.

En ce qui concerne l'estimation du temps de réponse de requête Mongodb, nous prenons en compte les paramètres les plus importants qui affectent l'exécution d'une requête, par exemple la taille des données, le nombre de fragments, les Entrées/Sorties et la bande passante réseau. Lorsque la réplication de données est décidée, seules les données populaires, c'est-à-dire ayant une fréquence d'accès élevée pendant une période de temps, et qui sont situées dans des nœuds de surcharge, sont répliquées. Ainsi, la charge de travail des nœuds est équilibrée suite à la réplication, ce qui affecte les performances du système. La stratégie de réplication proposée est également basée sur des localités géographiques [92] et de bande passante réseau [58] afin de rapprocher respectivement les nouvelles répliques des consommateurs de données et de réduire les coûts de communication lors de la réplication de données . En outre, le nombre de répliques est ajusté dynamiquement afin de réduire la consommation de ressources. Ainsi, les répliques de données les moins populaires sont supprimées. Enfin, afin de prendre en compte le bénéfice du fournisseur, nous estimons les revenus et les

dépenses du fournisseur lors de la réplication de données. Le fournisseur devrait avoir un réel bénéfice en considérant cette réplication.

3.4. La topologie considérée

Les fournisseurs de cloud offrent la possibilité de soumettre des requêtes aux locataires qui paient pour des services spécifiques. La plupart de ces requêtes requièrent des données situées sur des nœuds répartis sur des emplacements géographiquement répartis. Les fournisseurs de Cloud établissent souvent plusieurs installations dans des régions géographiques distinctes pour une multitude de raisons, y compris la fourniture de services qui couvre le monde entier . Par conséquent, choisir la bonne topologie du système de cloud (architecture) est l'une des décisions les plus importantes à prendre.

Dans cette thèse, nous considérons un cloud composé de plusieurs régions géographiques. Chaque région contenant un certain nombre de centres de données, comme le montre la figure 3.7

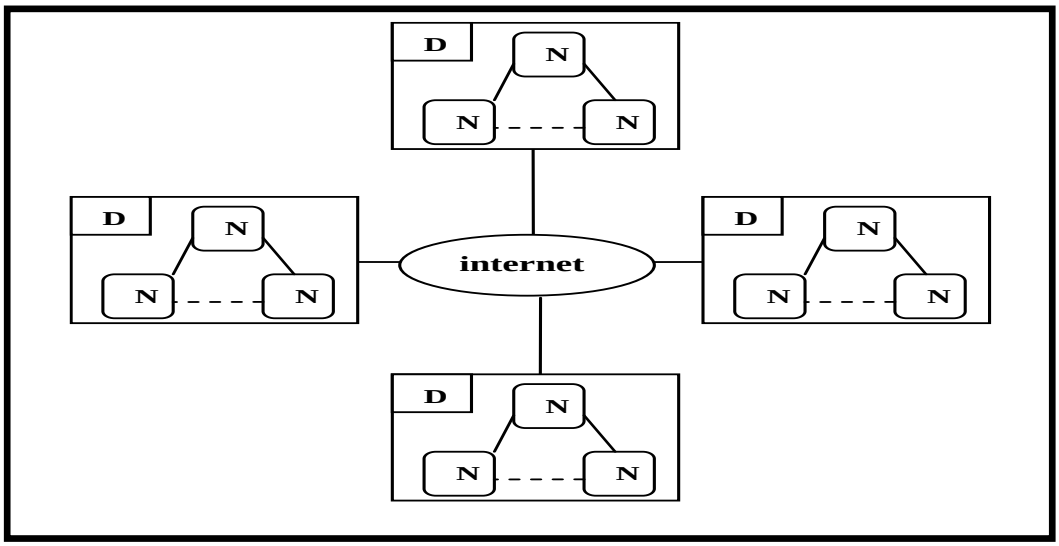


Figure 3.7. Architecture du système cloud adaptée.

Soit Dc_i un ensemble de centres de données noté $Dc_i = \{Dc_1, Dc_2, ..., Dc_n\}$, reliés entre eux par un réseau à faible bande passante e.g. internet. Chaque centre de données Dc_i contient un ensemble N de m nœuds considéré en tant que machines virtuelles avec $N = \{N_{i1}, N_{i2}, ... N_{im}\}$ communiquant via une large bande passante, comme le montre la figure 3.8. Plusieurs nœuds peuvent résider sur un même hôte physique et possèdent des caractéristiques spécifiques liées à la capacité de calcul, de mémoire, de stockage et de connectivité réseau afin de réaliser l'exécution des requêtes soumises.

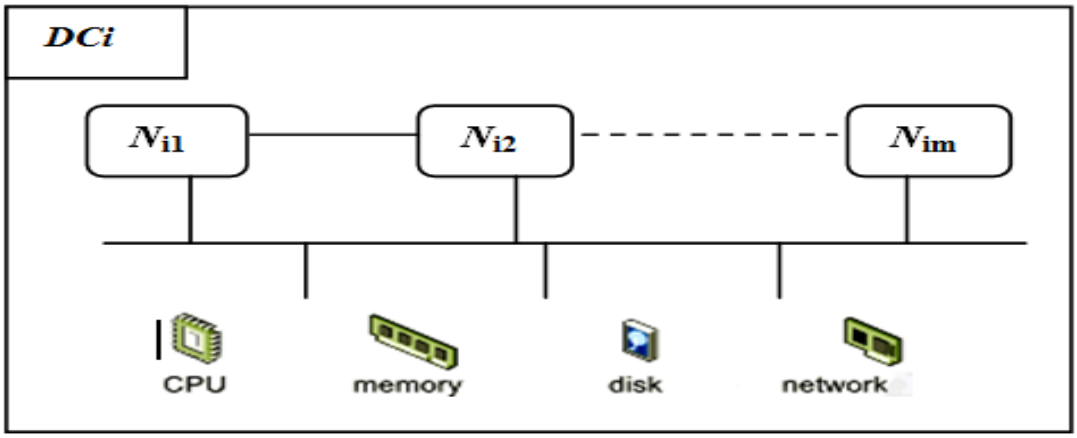


Figure 3.8. L'architecture de centre de données.

3.5. La stratégie proposée de la réplication de données

Du point de vue du locataire, il est essentiel que le temps de réponse d'une requête soit inférieur au seuil défini dans le SLA. Le fournisseur de cloud vise à satisfaire le SLA avec le montant maximum de profit. La stratégie que nous proposons se base sur deux modèles. le premier est le modèle de coût basé sur une estimation du temps de réponse et le deuxième est le modèle de coût économique utilisé pour l'estimation du profit du fournisseur.

Une caractéristique importante qui différencie le cloud et les environnements traditionnels, exemple : les grilles de données, c'est cette liaison entre le client et fournisseur. Cette liaison est matérialisée à travers le contrat SLA. Par la suite, si les termes du SLA ne sont pas respectés par le fournisseur, cela donne lieu à des pénalités. En effet, en cas de violation d'un accord de niveau de service, le fournisseur est tenu de payer une somme d'argent convenue au locataire [96]. Dans notre cas, lorsque le temps de réponse de la requête est supérieur au seuil défini dans le SLA, cela indique une violation du SLA. Il est donc important de noter que les pénalités jouent un rôle important dans l'économie du cloud.

Le mécanisme de réplication est déjà pris en charge dans les systèmes de geston de bases de données Nosql orientée document MongoDB. Il permet d'avoir un basculement automatique et un équilibrage de charge, alors que le but principal est d'obtenir une haute tolérance aux pannes en cas de défaillance d'un nœud primaire (maître). Cependant, les concepts économiques tels que le profit du fourbisseur ne sont pas pris en compte lors de la réplication.

Dans cette section, nous présentons une nouvelle stratégie de réplication dans Mongodb afin d'obtenir des meilleures performances tout en prenant en compte le profit d'un fournisseur de cloud. Toute stratégie de réplication de données doit répondre à quelques préoccupations clés dans le processus de décision pour la réplication afin de constituer une stratégie complète qui constitue des données appropriées pour la satisfaction d'un certain objectif.

Quand faut-il répliquer est le premier élément d'une stratégie de réplication de données. Répliquer trop tôt ou trop souvent entraînerait une utilisation inefficace des ressources et réduirait également les performances en raison de la réapparition fréquente des frais généraux de réplication. Si elle est faite trop tard ou trop tôt, elle peut être nuisible, car l'avantage de la réplication de données serait réduit.

Le deuxième rôle d'une stratégie de réplication de données est d'identifier correctement les données à répliquer. En particulier dans le traitement des requêtes de base de données, une requête peut nécessiter plusieurs relations au cours de son exécution. Une stratégie de réplication doit déterminer correctement si l'une de ces relations entrave le temps de réponse de la requête. L'identification incorrecte des éléments à répliquer ne ferait qu'entraîner des pertes de ressources sans réel bénéfice pour les performances. Dans la stratégie proposée, cette étape est gérée par le modèle d'estimation du temps de réponse

Le placement de la nouvelle réplique créée est aussi important que les questions précédentes. Un bon placement en temps convenable pour satisfaire aux exigences du système est essentiel à cette étape. Par conséquent, une réplication de données doit rapidement trouver un emplacement rentable afin de réduire le temps de réplication de données et ne pas perdre de temps à essayer de trouver le placement optimal.

Enfin, la suppression des répliques inutiles doit également être gérée par la stratégie de réplication de données. Au fil du temps, certaines répliques peuvent devenir inutilisées en raison de l'évolution de la demande dans le système de gestion des données. Pour libérer des ressources pour de futures répliques et augmenter la rentabilité en réduisant la consommation de ressources, ces répliques doivent être retirées du système au fur et à mesure que les requêtes passent dans le cloud.

Dans les sections suivantes, nous nous intéressons à ces problématiques en plus de de la rentabilité de la stratégie de réplication tout en satisfaisant les SLOs tels que les performances du systèmé.

3.5.1. La décision de la réplication

La décision de la réplication dépend de la vérification de deux conditions: (i) un temps de réponse est supérieur à un seuil défini de temps de réponse et (b) une réplication devrait être bénéfique pour le fournisseur.

Avant l'exécution de chaque requête Q, la stratégie estime le temps de réponse de Q (RespQ) et le compare avec le seuil de temps de réponse RespT défini dans SLA. Dans le cas où RespQ est supérieur à RespT, certains ensembles de données requis peuvent être répliqués afin de satisfaire l'objectif de performances. Puis, cette réplication n'est réellement déclenchée que si le fournisseur estime une certaine rentabilité suite à cette réplication. L'événement de réplication est déclenché comme indiqué dans l'algorithme de décision de réplication représenté sur la figure 3.9.

Algorithme: Décision de réplication

Début

1: *RespQ ← temps de réponse estimé d'une requête MongoDB Q*

2: ***Si*** *(RespQ > RespT)* ***alors***

3: *Sélectionner les données concernées par la réplication*

4: *NFound ← Noeud qui contient la nouvelle réplique // (RespQ < RespT)*

5: *p ← profit estimé par placement d'une nouvelle réplique*

6: ***Si*** *(NFound≠∅)* ***et*** *(p > 0)* ***alors*** *placement de réplique*

Fin

Figure 3.9. Algorithme de décision de réplication.

3.5.1.1. Modèle de coût pour le traitement des requêtes

Lorsque les locataires soumettent des requêtes au cloud, ils s'attendent à un délai de réponse en accord avec le SLA. Par conséquent, le modèle de coût évalue si une requête soumise peut être traitée avec un temps de réponse acceptable qui satisfait le SLA. Le modèle de coût proposé tient compte du parallélisme intra-opérateur et de la consommation de ressources des requêtes en se basant sur les études existantes [97]; [98].

- Estimation du temps de réponse d'une requête

La stratégie proposée se base sur l'estimation du temps de réponse afin de décider si les données doivent être répliquées ou non pour satisfaire aux exigences du locataire.

La parallélisassions des tâches est l'une des caractéristiques les plus importantes de l'utilisation d'un cloud. Nous bénéficions de plusieurs études de bases de données relationnelles existantes, par exemple ([97];[98]). Traditionnellement, on distingue deux types de parallélisme: intra-opérateur et inter-opérateur afin de traiter une grande quantité de données et d'améliorer les performances. L'estimation du temps de réponse d'une requête est basée sur la consommation de ressources lors de l'exécution de cette requête.

Dans notre travail, nous nous concentrons sur l'estimation du temps de réponse pour une requête qui n'a pas d'opérations dépendantes. En fait, les requêtes de recherche classiques dans Mongodb sont sans jointures comme la plupart des systèmes NoSQL. En conséquence, nous nous intéressons seulement au parallélisme intra-opérateur qui consiste en l'exécution parallèle de plusieurs opérateurs lancés sur plusieurs sites cloud. Ensuite, le résultat consiste en la combinaison des résultats générés de chaque site. Il peut y avoir une variation de réponse entre l'exécution d'un opérateur sur des fragments différents en raison des ressources matérielles individuelles, de la mémoire, du processeur ... etc.

Supposons qu'une requête Q d'un client est soumise à un cloud contenant plusieurs centres de données, chacun contenant un certain nombre de nœuds. Supposons maintenant que Q est délégué à un nœud donné qui doit renvoyer un résultat au

locataire. *Q* peut être exécuté sur plusieurs nœuds et plusieurs fragments. Soit *Q* une requête locataire composée de plusieurs sous-requêtes $Q = \{Q_1, Q_2, ..,.,Q_i\}$ qui sont exécutées en parallèle sur k fragments. Soit Q_i une sous-requête qui nécessite une collection C_n. Supposons que la collection C_n soit divisée en *n* morceaux comme $C_n = \{C_{n,1}, C_{n,2}, ..., C_{n,l}\}$, avec des répliques sur les sites *j*. Nous nous référons à la sous-requête Q_i qui est exécutée sur *k* fragments en parallèle par $Q^{j}_{i,k}$.

Dans ce cas, le temps de réponse estimé de la requête soumise est défini par : (i) le temps estimé le plus long pour $Q^{j}_{i,k}$ exécuté sur *k* fragments, où *k* est le nombre de fragments, (ii) le transfert de données à partir d'un fragment à un autre et (iii) la production des résultats. Le temps de réponse pour l'exécution de la requête RespQ est indiqué dans l'équation suivante [97]:

$$RespQ=Max_k[Resp(Q^{j}_{i,k})] + T_{Tr} + T_{Pr} \qquad (1)$$

Où T_{Tr} est le temps nécessaire pour fournir les résultats. En effet, les résultats intermédiaires sont transférés de tous les fragments concernés vers une destination finale. Ensuite, nous devrions produire le résultat final, par exemple, un opérateur d'union est appliqué pour regrouper des collections fragmentées horizontalement. Nous nous référons au temps requis pour stocker et produire les résultats de sortie par T_{Pr}.

Pour calculer le temps de réponse de l'opérateur exécuté sur chaque fragment en parallèle $Resp(Q^{j}_{i,k})$, il est nécessaire d'évaluer la quantité de temps apportée par les ressources de calcul, y compris les E/S disque, l'espace tampon, le CPU et le temps de communication comme indiqué dans l'équation suivante:

$$Resp(Q^{j}_{i,k}) = Max([T^{CPU}(Q^{j}_{i,k}) + T^{I/O}(Q^{j}_{i,k})], T^{Com}(Q^{j}_{i,k})) \qquad (2)$$

a. Estimation du temps CPU

Le temps lié à la CPU dépend du temps nécessaire pour passer les instructions nécessaires au traitement d'une quantité d'instructions. L'estimation d'un temps CPU pour la sous-requête $Q^{j}_{i,k}$ comme représenté dans l'équation (3):

$$T^{CPU}(Q^{j}_{i,k}) = T_{CPU} * \#Inst * (1 + a) \qquad (3)$$

Avec T_{CPU}, est le temps d'une instruction CPU, et $a>0$ est un facteur de pondération prenant en compte les capacités matérielles du processeur, par exemple, le débit de traitement et les capacités de mise en cache sur N_{ij} qui influencent les performances de la requête.

b. Estimation du temps I/Os

Q peut requérir le document d_l d'un nœud local N_{ip} $(1 \le i \le m, 1 \le p \le n)$et / ou d'un nombre *r'* de documents distants distribués sur des nœuds distants (d_r peut constituer des résultats intermédiaires). Soit Sd_l la taille du total des documents locaux (en octets) requis sur N_{ip}. Sd_r la taille des documents distants ($\in$ nœud distant) requis pour le traitement Q (en octets), IO_r and IO_{ip} les débits moyens des disques d'E /S sur un nœud distant et N_{ij} respectivement (en octets / s), y compris l'accès aléatoire et séquentiel à

une page, où r = [1 ... j]. Ainsi, le coût d'E/S estimé de $Q^{j}_{i,k}$ (en sec) sur N_{ip} qui nécessite des documents locaux et des documents distants est donné par la formule suivante:

$$T^{I/O}(Q^{j}_{i,k}) = \sum_{r=1}^{n} (Sd_r / IO_r) + Sd_l / IO_{ip} \qquad (4)$$

c. Estimation du temps de communication

Une requête MongoDB peut nécessiter des documents distants. En conséquence, les coûts de communication doivent être pris en compte. Cela représente le temps passé sur la communication entre les nœuds. Nous devons estimer le temps de migration des documents distants d_r vers le nœud local où r = [1 ... j]. Soit Sd_r la taille de d_r et BN_{ip} la bande passante réseau moyenne de N_{ip} (en octets / s). L'estimation d'un temps de transfert pour $Q^{j}_{i,k}$ qui nécessite *n* documents distants d_r est représentée dans l'équation (5):

$$T^{Com}(Q^{j}_{i,k}) =) = \sum_{r=1}^{n} Sd_r / BN_{ip} \qquad (5)$$

3.5.1.2. Modèle économique pour le traitement de requêtes

Ce modèle économique estime la rentabilité du fournisseur en évaluant le coût monétaire de l'exécution de chaque requête. Dans la stratégie que nous proposons, l'estimation du bénéfice proposée est utilisée comme critère de décision dans le processus de réplication de données.

La plupart des stratégies de réplication existantes dans le cloud visent à obtenir de meilleures performances sans tenir compte des profits du fournisseur. Dans notre stratégie, la réplication d'une collection est considérée uniquement si un fournisseur a un profit économique. Dans ce contexte, nous devons estimer les revenus et les dépenses du fournisseur lors de l'exécution d'une requête et en considérant la nouvelle réplique à créer. Du point de vue du fournisseur, le bénéfice est estimé comme indiqué dans l'équation (6):

$$Profit = revenus - dépenses \qquad (6)$$

Tous les fournisseurs cloud visent à maximiser les revenus payés par les utilisateurs pour le traitement de leurs requêtes. Cela justifie l'exécution de la requête en parallèle. Habituellement, le but de tout fournisseur de services cloud est d'obtenir beaucoup plus de gains, c'est-à-dire des bénéfices. En conséquence, les dépenses du fournisseur doivent être minimisées.

En cas de violation de SLA, le fournisseur paie une pénalité au locataire. En conséquence, un défi intéressant consiste à éviter le coût de pénalité payé par le fournisseur à l'utilisateur qui est dû à la non satisfaction d'un SLO donné. La rentabilité des fournisseurs de cloud revient donc à la minimisation des pénamlités provoquée par la violation de l'accord SLA.

a. Estimation des revenus

Un locataire n'est pas facturé pour le nombre de répliques requis lorsque sa requête est exécutée, c'est-à-dire que ce processus est transparent pour le locataire. Cependant,

un locataire doit périodiquement payer le fournisseur pour le temps de calcul, le coût opérationnel et les ressources allouées lors du traitement de sa requête. Ainsi, le fournisseur devrait fixer le prix du service en fonction de toutes ses dépenses afin de procurer un gain raisonnable tout en évitant une perte de profit.

Dans cette sous-section, nous estimons un revenu par requête. Cela dépend du taux d'arrivée maximum des requêtes *Max_AR* et de la durée de la période de facturation *BP*, déjà spécifiée dans le SLA. Cela dépend également du montant du loyer que le locataire paie au fournisseur de cloud pour les services acquis au cours d'une BP. Le revenu estimé pour l'exécution de Q est indiqué dans la formule suivante:

$$REV_Q = Rent / Max\_AR * BP \qquad (7)$$

b. Estimation des dépenses

L'évaluation d'une requête Q génère des coûts opérationnels impactés par différents paramètres. Soit *Cost_t* le coût par unité de temps *u* pour utiliser un nœud alloué à l'évaluation d'une requête Q [46]. *T_NeedQ* est le temps total estimé nécessaire pour évaluer la requête Q. *Nodes_Need* est le nombre des nœuds requis pour évaluer une requête Q, *Netw_cost* est le coût de l'utilisation de la bande passante y compris la migration des données d'un nœud à l'autre lors de la création d'une réplique, *Stor_cost* est le coût nécessaire pour stocker les répliques sur les disques, *Softw_price* le prix de la licence logicielle, et *Tenants_Penalty* les pénalités payées par le fournisseur aux locataires lorsqu'un ou plusieurs SLO_i n'est /ne sont pas satisfait (s). L'estimation des dépenses du fournisseur lors de l'évaluation de Q est indiquée dans la formule suivante:

$$Q\_Expense_t = \Sigma_1^{Nodes\_Need}(T\_NeedQ * Cost\_t) + Netw\_cost + Stor\_cost + Softw\_price + \Sigma_1^{n} Tenant\_penalty \qquad (8)$$

3.5.2. La sélection des données à répliquer

Dans le cas où le temps de réponse estimé de la requête Q (*RespQ*) est supérieur à *RespT*, la stratégie de réplication doit sélectionner les données qui sont concernées par la réplication.

Nous nous sommes basés sur la fréquence d'accès de chaque donnée pendant une période de temps. Cela consiste à sélectionner les ensembles de données ayant le plus haut degré de popularité. Selon [30], la fréquence d'accès de chaque fichier est analysée en considérant le nombre de fois où il y a eu accès au fichier pendant un intervalle de temps particulier. Le fichier ayant la plus grande valeur de fréquence d'accès est sélectionné en tant que fichier populaire. Le degré de popularité peut être calculé en utilisant l'équation suivante [32]:

$$pd_k = \sum_{ti=ts}^{tp} (an_k(t_i, t_{i+1}) * w(t_i, t_p)) \qquad (9)$$

Où t_s, t_p, présentent respectivement l'heure de début et l'heure actuelle et $an_k(ti, t_{i+1}))$ présente le nombre d'accès pendant l'intervalle de temps t_i à $ti + 1$. $w(t_i, t_p)$ présente un temps basé sur La fonction d'omission (w) qui est définie sur le domaine du temps, avec des valeurs dans l'intervalle [0, 1]. Elle est utilisée pour calculer le degré de popularité

d'un bloc de données à l'instant t_p en se basant sur la fréquence d'accès à l'heure de début t_s. Lors de l'analyse de la popularité des données, seules les données populaires sont répliquées. Par conséquent, il n'est pas nécessaire de créer des répliques pour tous les ensembles de données, en particulier pour les fichiers moins consultés (données non populaires).

Le deuxième critère sur lequel nous nous sommes basés est la charge de travail d'un nœud. Nous considérons la réplication de données situées sur des nœuds surchargés. En outre, certaines données particulières sont consultées beaucoup plus fréquemment que d'autres. Donc, la charge sur les nœuds qui hébergent les données sera augmentée, ce qui peut affecter les performances du système. Pour résoudre ce problème, nous sélectionnons les données situées sur un nœud surchargé et les nous plaçons dans un autre nœud non surchargé. Une charge de travail d'un nœud donné renseigne sur la quantité de travail à effectuer. La définition d'un nœud surchargé peut être donnée comme suit: pour chaque nœud (N_{ip}), nous pouvons avoir sa charge notée *load* (N_{ip}), qui dépend de deux paramètres: (i) la capacité de calcul c et (ii) la capacité du réseau b . Dans le cas où nous avons trouvé que *load* (N_{ip}) est plus grand que *load_threshold*, nous déclarons que le nœud actuel N_{ip} est surchargé. Par conséquent, nous répliquons des ensembles de données fréquemment utilisées dans ce nœud afin de réduire la charge de travail de N_{ip} comme indiqué dans l'algorithme représenté sur la figure 3.10.

Algorithme: Sélection de données

Début

1: $RespQ_{ip}$ ← temps de réponse estimé de Q sur N_{ip}
2: ***Si*** *($RespQ_{ip}$ >RespT)* ***alors***
3: ***Si*** *load(N_{ip})> load_threshold* ***alors***
4: { Sélectionnez les données situées dans N_{ip} où Pd >=Max_{PD}
5: trouver le nœud approprié pour placer les répliques
6: déclencher la réplication }

Fin

Figure 3.10. Données concernées par la réplication.

3.5.3. Le placement des nouvelle répliques

Lorsque le temps de réponse estimé pour une requête, soumise sur $N_{ij} \in DC_i$, est supérieur au seuil *RespT*, et après avoir sélectionné les données concernées par la réplication, nous devons trouver le nœud approprié pour contenir la nouvelle réplique tels que *RespQ <RespT*. Sur la base de la topologie du cloud présentée dans la section (3.4), nous profitons de la localisation au niveau de la bande passante réseau pour placer des répliques. C'est-à-dire qu'il faut placer les nouvelles répliques sur des nœuds qui ont une bonne bande passante avec N_{ij}. Par conséquent, la recherche du nœud pour placer les nouvelles répliques doit être effectuée dans le centre de données local. Le nœud

sélectionné doit également avoir suffisamment d'espace de stockage et ne pas être surchargé. Ceci constitue la première condition.

Algorithme: Emplacement de réplique

Début

$DC= \{Dc_1, Dc_2, ...Dc_n\}$, *Let* $N=\{N_{i,1}\ N_{i,2} ..... N_{i,m}\} \in Dc_i$,

$RespQ_{ip}$ *le temps de réponse estimé sur* $N_{i,p}$

Supposer que $RespQ_{ip} > RespT$ *on* $N_{i,p}$

1: found ← flase

2: ***Pour*** *(k=1, k<m, k++) //k!=p*

3: ***Début***

4: ***Si****(*$RespQ_{ik} < RespT$*)* ***alors*** $NFound=N_{ik}$

5: found ← true; Aller à 14

6: ***Fin***

7: ***Si*** *(found=false)* ***alors***

8: ***Début***

9: La recherche de Dc_k *avec la meilleure bande passante à DCi*

// supposons que les nœuds de $N_{kk'}$ *de* DC_k

10: ***Si*** *(*$RespQ_{kk'} < RespT$*) et (*$N_{kk'}$*.freespace> sizeof(d)) et (overload* $N_{kk'}$ *<LoadT)* ***alors*** *found ← true; NFound=*$N_{kk'}$ *Aller à 14*

11: ***Sinon*** *Recherche de* Dc_k *au meilleur prix de stockage*

12: ***Si****(*$RespQ_{kk'} < RespT$*) et (*$N_{kk'}$*.freespace > sizeof(d)) et (overload* $N_{kk'}$ *<LoatT)* ***alors*** *found ← true; NFound=*$N_{kk'}$ *Aller à 14*

13. ***Fin***

14: ***Si*** *found= false* ***alors*** *aller à 17*

15: ***Si*** *(NFound* $\neq \emptyset$ *)* ***alors*** *Estimer le profit lors de l'exécution de Q*

16: ***Si*** *(Profit >0)* ***alors*** *Placez la nouvelle réplique sur NFound*

17: ***Fin***

Figure 3.11. Algorithme de placement des nouvelle répliques.

La deuxième condition consiste à ce que le fournisseur doit également avoir un profit lors de la réplication de données sur ce nœud, c'est-à-dire que le bénéfice du fournisseur est vérifié lors de l'exécution d'une requête qui utilise la réplique concernée. Enfin, s'il y a un réel bénéfice, la nouvelle réplique est réellement placée sur un nœud de placement de réplique *NFound*. Si l'une de ces deux conditions n'est pas satisfaite pour tous les nœuds du centre des données locales, la recherche est effectuée dans d'autres centres de données afin qu'on puisse trouver le nœud approprié. Premièrement, nous recherchons le centre de données qui a la meilleure bande passante avec le centre de données qui contient N_{ij}. Si l'objectif de temps de réponse n'est pas atteint, la recherche est appliquée sur le centre de données qui offre les meilleurs prix de stockage. Bien entendu, le nœud recherché devrait avoir la possibilité de stockage pour contenir une nouvelle réplique et il ne devrait pas être surchargé, c'est-à-dire que la charge devrait être inférieure à la charge de seuil dans le SLA. voir figure 3.11.

3.5.4. La suppression de répliques

Les fournisseurs de cloud visent à satisfaire les exigences des locataires tout en maximisant ses avantages. Pour cet objectif ultérieur, il convient de minimiser les dépenses lors de l'exécution des requêtes de clients. Une gestion des ressources élastique traite du retrait de toutes les ressources non requises. Par exemple, les répliques qui ne sont pas utiles doivent être supprimées. Cela réduit les dépenses du fournisseur et augmente le bénéfice du fournisseur.

Soit Q une requête de locataire soumise dans DC_i. Pour chaque nœud N_{ij}, nous sauvegardons l'historique de l'accès aux données (répliques). Lorsque le temps de réponse estimé sur N_{ip} est significativement plus lent que *RespT*, c'est-à-dire (*RespQ*<*ß*.RespT), le retrait de réplique est préconisé. Pour ce faire, nous devrions également estimer le degré de popularité de chaque segment de données. Nous sélectionnons les répliques ayant le plus faible degré de popularité Popd(Ri), comme indiqués dans l'algorithme de suppression des répliques (figure 3.12).

Algorithme: Suppression de répliques

Debut

1: $RespQ_{ip}$ ← temps de réponse estimé de l'exécution de la requête Q soumise dans Dc_i

2: ***Si*** *($RespQ_{ip}$ <<β. RespT)* ***alors***

3: ***Pour*** *chaque réplique dans les centres de données Dc_i* ***faire***

4: Analyser et enregistrer l'historique récent de l'accès aux données

5: Calculer le degré de popularité Popd(Ri) de répliques Ri

6: ***Si*** *(Popd(fi) <pd_T)* ***alors*** *Supprimer la réplique Ri*

7: ***Fin***

Figure 3.12. Algorithme du suppression des répliques.

La suppression des répliques inutiles permettra un gain au niveau de l'espace de stockage matériel. Rappelons que nous spécifions un seuil de degré de popularité noté *pdT* sur le *SLA*. Rappelons également que nous devrions conserver un nombre minimum de répliques afin de garantir une disponibilité minimum.

3.6. Conclusion

Dans ce chapitre, nous avons présenté une nouvelle stratégie de réplication de données pour Mongodb, un système de gestion de bases de données NoSQL orienté documents. L'objectif principal de la stratégie proposée est de satisfaire l'exigence de performance pour l'utilisateur tout en tenant compte du profit économique du fournisseur. La réplication est déclenchée uniquement si le temps de réponse estimé d'une requête soumise est supérieur à un seuil de temps de réponse défini préalablement dans le SLA. Ensuite, la réplication doit être rentable pour le fournisseur lorsqu'une réplication est considérée. Notre stratégie est basée sur: (i) un modèle de coût qui permet d'estimer le temps de réponse d'une requête avant d'exécuter celle-ci, et (ii) un modèle de coût économique qui permet d'estimer à la fois les revenus et les dépenses du fournisseur lors de l'exécution de cette requête afin de ne répliquer que si cette réplication est profitable pour le fournisseur. Nous avons commencé par déterminer le moment de la réplication puis les données à répliquer. Puis, une fois que la réplication de données est considérée, nous avons proposé un algorithme de placement d'une nouvelle réplique. Nous avons également traité la réplication de cette réplique de manière dynamique à travers l'ajustement dynamique du nombre de répliques nécessaire à la satisfaction du SLA et la satisfaction du profit du fournisseur.

CHAPITRE 4 : EVALUATION DES PERFORMANCES

CHAPITRE 4 : EVALUATION DES PERFORMANCES 73
4.1. Introduction 74
4.2. Environnement de simulation 75
4.3. Mise en place d'YCSB 75
4.4. Evaluation du mécanisme de réplication de données existant dans MongoDB 76
4.5. Évaluation de la stratégie proposée 79
4.6. Discussion 80
4.7. Conclusion 83

4.1. Introduction

Dans le chapitre précédent, nous avons présenté les différents aspects de la stratégie de réplication de données proposée. Une autre étape cruciale pour proposer une nouvelle stratégie de réplication de données consiste à la valider à travers une évaluation expérimentale. Bien sûr, l'obtention de résultats reproductibles revêt une importance capitale dans l'évaluation des performances.

Dans ce chapitre, l'évaluation expérimentale de la stratégie proposée peut être résumée en trois étapes: (i) l'évaluation des performances de Mongodb en se basant sur le mécanisme de réplication de données déjà existant, (ii) l'évaluation des performances de Mongodb en adaptant notre stratégie de réplication de données, et (iii) la comparaison des résultats obtenus. Nous terminons ce chapitre par une analyse des résultats obtenus.

4.2. Environnement de simulation

Afin d'évaluer l'impact de notre stratégie de la réplication de données sur les performances du Mongodb, nous avons utilisé le Yahoo Cloud Serving Benchmark (YCSB) [99]. YCSB est un outil développé en open source pour évaluer les performances de différentes bases de données du service Cloud, y compris les systèmes Nosql par exemple: MongoDB [94]. Cette étude a été réalisée dans un environnement dont les spécifications et les paramètres sont indiqués dans le tableau (4.1).

Setting	Value
OS	ubuntu 16.04 LTS
Word length	64-bits
RAM	8 GB
Hard Disk	1TB
CPU Speed	1.70 GHz
Core	5
YCSB version	0.5.0
MongoDB version	3.2.16
netbeans IDE version	8.0.2

Tableau 4.1. Spécifications et paramètres de la machine.

4.3. Mise en place d'YCSB

Pour la mise en place de YCSB, il faut d'abord installer Java, Maven et Git dans notre système, ensuite il faut tester le fonctionnement des dépendances:

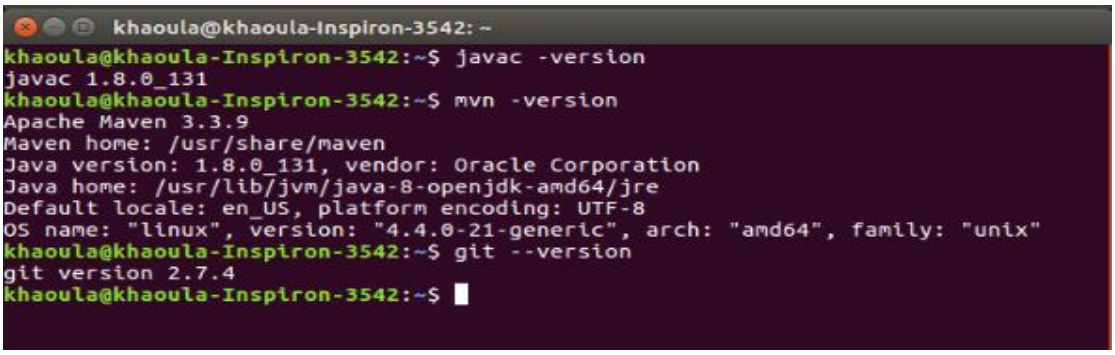


Figure 4.1: Les tests du fonctionnement des dépendances.

Nous avons téléchargé et compilé le source YCSB, et nous avons obtenu les résultats suivants :

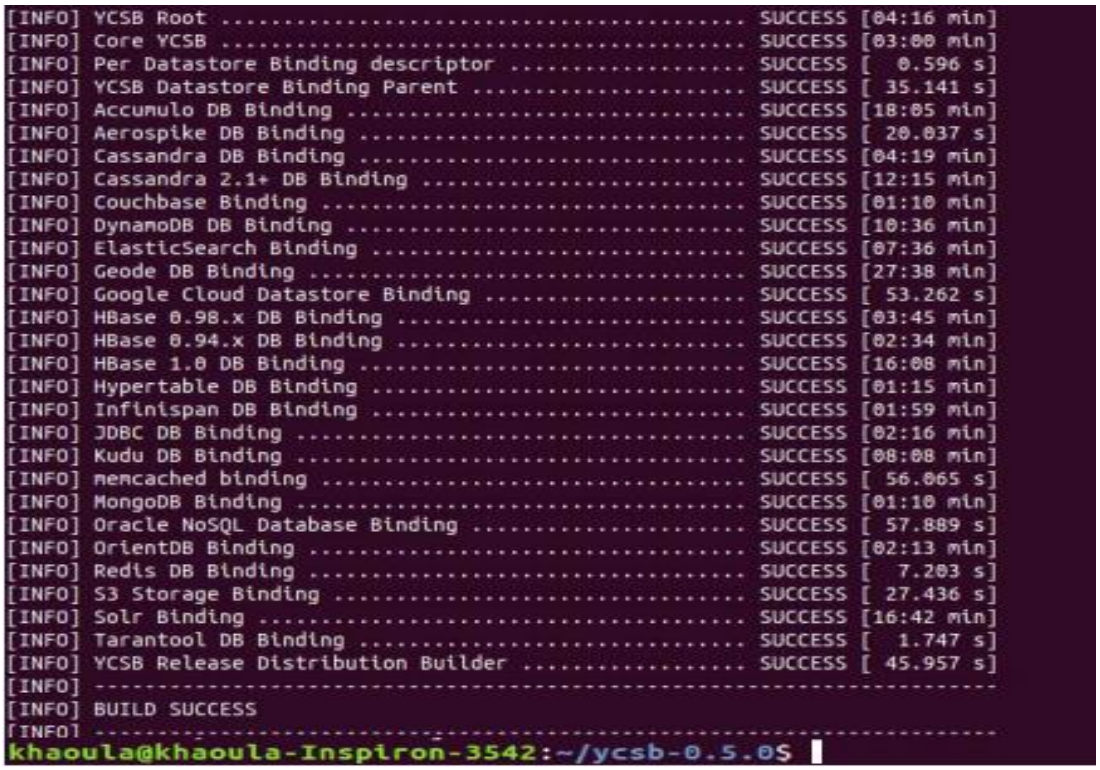
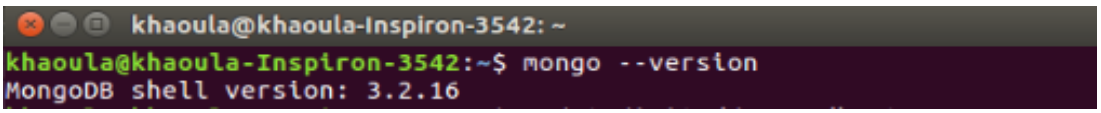


Figure 4.2. Phase de téléchargement et compilation du source YCSB.

Après le téléchargement et l'installation du Mongodb sur la machine, Le lancement et la vérification de la version de MongoDB sont fait par :



4.4. Evaluation du mécanisme de réplication de données existant dans MongoDB

Pour l'analyse expérimentale, nous avons utilisé le YCSB (Yahoo!Cloud Serving Benchmark), qui est un framework Open Source largement utilisé pour l'évaluation et la comparaison des différents types de systèmes de données actifs (y compris les bases de données NoSQL). Le benchmark se compose de deux éléments : un générateur de données et un ensemble de tests de performances pour évaluer les opérations de lecture et mise à jour. Chacun des scénarios de tests s'appelle une charge de travail.

Dans le contexte de YCSB, on peut définir un ensemble de charges de travail qui définissent un point de référence de base (benchmark) pour les systèmes cloud. Dans notre expérimentation, nous avons utilisé six charges de travail différentes qui sont configurables, chacune comprenant 1000 opérations sur des enregistrements de 1 Ko (10 champs, 100 octets chacun et clé par défaut) déjà chargées dans une base de données. Le tableau 4.2 montre la description des charges de travail testées.

Workload	Operations
A	50% Read, 50%Write
B	95%Read, 5%Write
C	100%Read
D	95%Read, 5%insert
E	95%Scan, 5%insert
F	50%Read, 50%RMW

Tableau 4.2. Description des charges de travail.

Nous prenons en compte deux métriques: le débit et temps de réponse total (throughputs and total response time). Le débit peut être défini comme une mesure à utiliser pour déterminer la performance d'un système de base de données, il présente le nombre des transactions produites au fil du temps lors d'un test. D'autre part, le temps de réponse est mesuré à partir du moment où un utilisateur envoie une requête jusqu'à ce que l'application indique que la demande est terminée (la réponse est fournie)[31].

Les deux figures suivantes présentent les résultats de chargement et d'exécution de la charge du travail A (workload A):

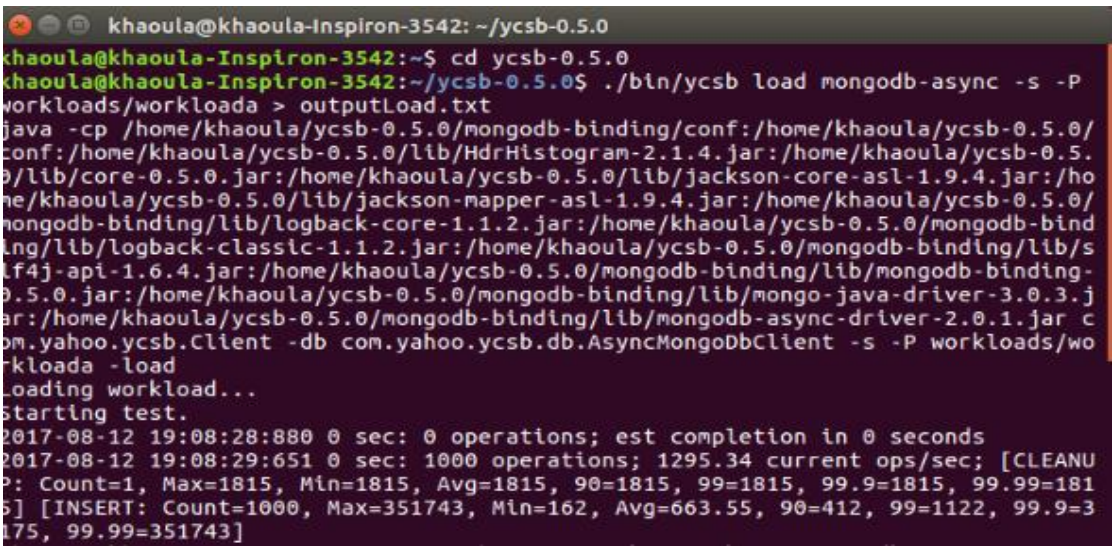
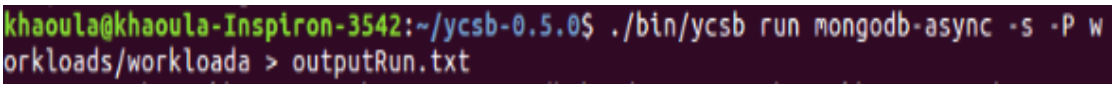


Figure 4.3. Résultats du chargement (load) de la charge du travail A.

Après le chargement (load) de toutes les charges de travail (A, B, C, D, E, F), on fait la même chose pour l'exécution de ces charges du travail :



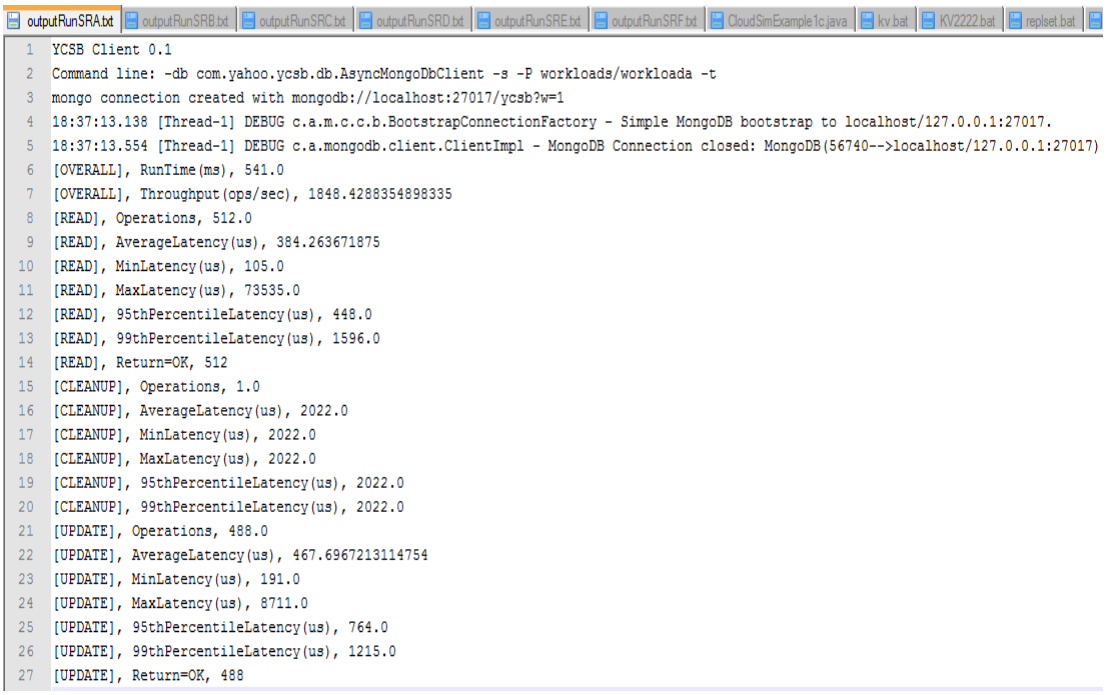


Figure 4.4. Résultat d'exécution (run) de la charge du travail A.

La figure 4.4 présente les résultats obtenus lors de l'exécution de chaque charge de travail décrite dans le tableau 4.3.

Workload	Throughput (Ops/sec)	Execution time (ms)
A	1848.42	541.0
B	2293.57	436.0
C	2386.63	419.0
D	2352.94	425.0
E	870.32	1149.0
F	1610.30	621.0

Tableau 4.3. Résultats obtenus lors de l'exécution de chaque charge de travail

La figure 4.5 représente les résultats de l'exécution de chaque charge de travail. Les résultats ont été obtenus en prenant en compte deux paramètres: le temps d'exécution et le débit. Nous remarquons que les performances de Mongodb sont plus meilleures lors de l'exécution d'opérations en lecture seule (charge de travail C). Cela renforce notre choix d'appliquer notre stratégie aux opérations en lecture seule (read only operation), d'autant plus que l'objectif est d'améliorer les performances grâce à la réplication de données. Par conséquent, nous évaluons la stratégie de réplication

proposée lorsque nous traitons des opérations en lecture seule. C'est le cas pour les applications OLAP comme dans notre cas.

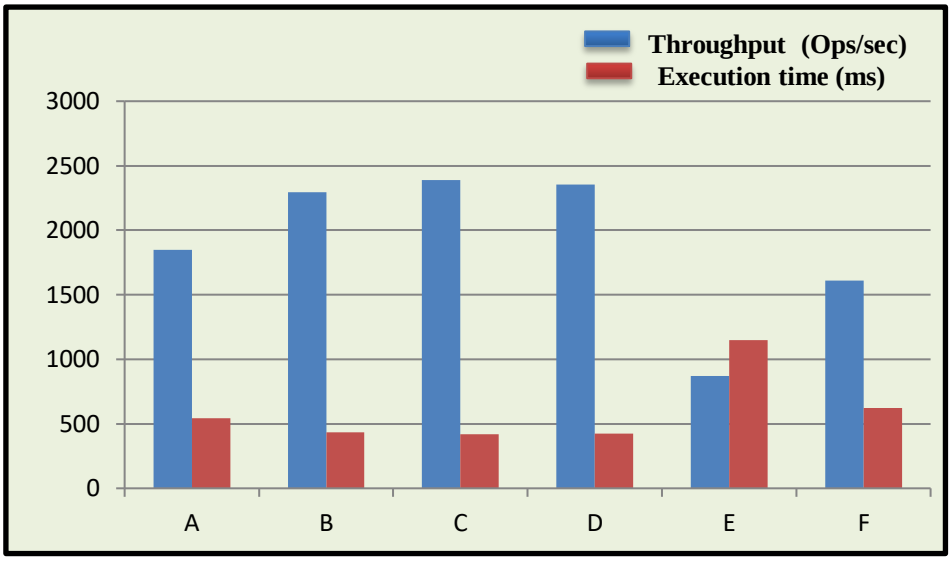


Figure 4.5. Les changements des Résultats de l'exécution des charges de travail.

4.5. Évaluation de la stratégie proposée

Notre objectif principal est de montrer que la stratégie proposée améliore les performances du système tout en prenant en compte le profit du fournisseur. Afin d'évaluer notre stratégie de réplication proposée dans Mongodb, nous avons utilisé les paramètres qui sont indiqués dans le tableau 4.1 (section 4.2). Pour interagir avec Mongodb, nous avons utilisé sa propre API Java Mongo. Ensuite, pour avoir une connexion java mongo, nous avons ajouté des fichiers *.jar* au projet netbeans IDE (version 8.0.2).

```
22  public class testperf {
23
24      /**
25       * @param args the command line arguments
26       */
27      public static void main(String[] args) {
28                      Mongo mongo = null;
29          try {
30              mongo = new Mongo ("localhost", 27017);
31          } catch (UnknownHostException ex) {
32              Logger.getLogger(testperf.class.getName()).log(Level.SEVERE, null, ex);
33          }
            DB db = mongo.getDB("evaluation1");
35          DBCollection collection = db.getCollection("mongoperftest1");
36          long start;
37          long end;
38          start = System.currentTimeMillis();
39          for (int i = 0; i <10000;i++) {
40              collection.insert(new BasicDBObject("_id", i), WriteConcern.SAFE);
41          }
42          end = System.currentTimeMillis();
43          long result= end - start;
44          System.out.println("insert: " + result + "ms");
45        if (result >5500) {
46          System.out.println("necessary replication: /n trigger replication");
47          }
48         mongo.close();
49
```

Figure 4.6. Exemple d'un code source utilisé pour l'expérimentation

Après l'implémentation de la stratégie de réplication proposée, les premières expérimentations consistent à soumettre des requêtes de lecture seules afin de simuler différentes charges de travail.

Nous avons spécifié un seuil de temps de réponse de 5500 ms pour chaque requête. Nous supposons que cette valeur est le résultat d'une négociation entre le fournisseur et le client, après une période de test. Ensuite, la réplication devrait être rentable pour le fournisseur. Au cours des expérimentations, nous traitons deux mesures importantes: (i) le temps de réponse et (ii) le nombre de réplications.

Nous notons le temps de réponse obtenu avec la lecture des 10, 100, 1000, 10000, 100000 et 1000000 enregistrements dans une collection. Les temps de réponse obtenus avec la stratégie de réplication déjà existante et la stratégie proposée sont illustrés dans la figure 4.7. Rappelons qu'avec la stratégie proposée, l'événement de réplication est déclenché uniquement si le temps de réponse estimé est supérieur à un seuil de temps de réponse.

4.6. Discussion

La figure (4.7) montre la variation du temps de réponse avec les deux stratégies de réplication comparées tout en faisant varier la charge de travail. Avec un nombre réduit de requêtes, un nombre légèrement plus important de répliques est créé par la stratégie de réplication existante dans MongoDB. Mais en revanche, la stratégie comparée ne prend pas en compte les profits du fournisseur. Un nombre de réplications moindres est requis avec notre stratégie puisque le temps de réponse est inférieur au seuil du temps de réponse. Ensuite, il y a une disparité marquée lorsque la charge de travail augmente. Peu à peu, lorsque nous avons plus de charges, plus de réplications se produisent par la suite avec notre stratégie afin d'éviter la violation SLA. Cela est dû au fait qu'une réplication est déclenchée chaque fois que le temps de réponse estimé est supérieur à un seuil de temps de réponse. Cela répond à l'objectif des fournisseurs de cloud. En fait, le bénéfice du fournisseur devrait être maximisé grâce à la minimisation de la violation SLA.

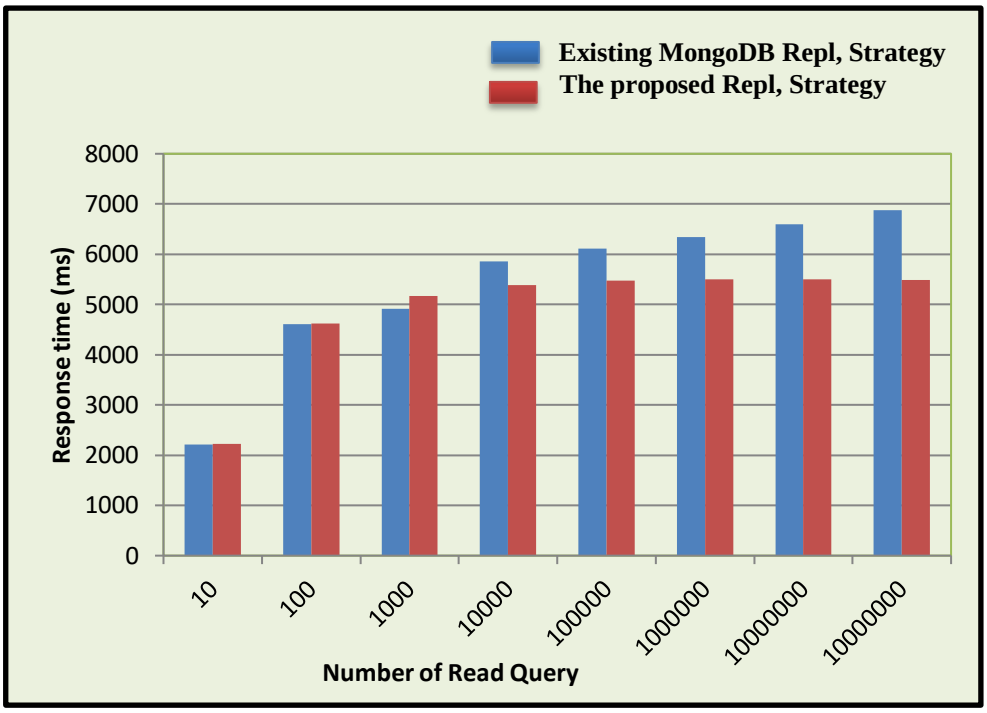


Figure 4.7. Résultats obtenus.

Concernant le temps de réponse, celui produit par la stratégie comparée est légèrement meilleur lorsqu'il s'agit de 1000 requêtes de lecture seules. Cela correspond à un plus grand nombre de répliques créées par la stratégie de réplication existante dans MongoDB. Mais en revanche, la stratégie comparée ne prend pas en compte les profits du fournisseur. D'un autre côté, un nombre moindre de réplications sont nécessaires avec notre stratégie puisque le temps de réponse est inférieur au seuil permis du temps de réponse. Puis, progressivement, lorsque nous avons plus de charge avec la stratégie proposée, plus de réplications se produisent par la suite afin d'éviter les violations du SLA.

La figure 4.7 montre les valeurs de temps de réponse obtenues lors de l'expérimentation avec les deux stratégies de réplication de données. Avec un petit nombre de requêtes, nous avons obtenu un temps de réponse similaire avec les deux stratégies. Le temps de réponse de la stratégie proposée est meilleur lorsqu'il s'agit de plus de 100 requêtes de lecture seules. Avec une charge élevée, il est clair que notre stratégie de réplication produit des temps de réponse moins importants. Les valeurs de temps de réponse de la stratégie proposée devraient être inférieures au seuil du temps de réponse, sinon, une réplication de données est déclenchée. Cela explique pourquoi nous avons des réplications plus importantes avec la stratégie proposée (voir figure 4.8).

D'un autre côté, chaque fois que la stratégie de réplication comparée dépasse le seuil de temps de réponse, nous avons des violations du SLA. En conséquence, le fournisseur devrait payer une pénalité au locataire lorsqu'il n'y a pas de pénalité à payer avec la stratégie proposée. En conséquence, le bénéfice du fournisseur est maximisé.

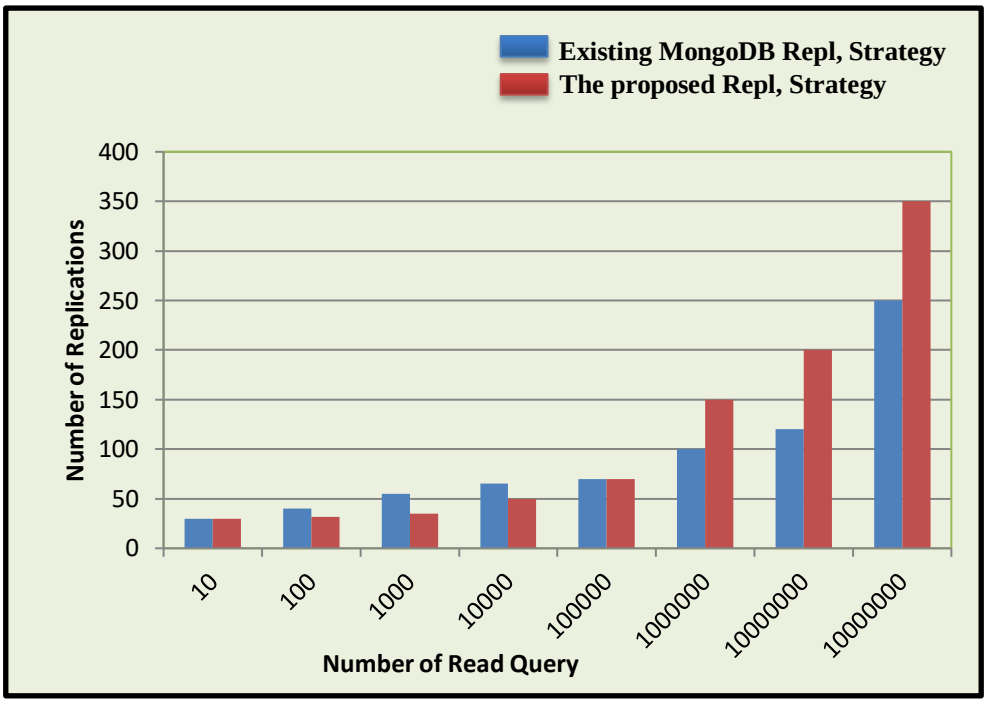


Figure 4.8. Nombre de réplications.

Le profit du fournisseur est également maximisé grâce à la gestion élastique des ressources. Par conséquent, la stratégie proposée évite les réplications inutiles en supprimant les répliques lorsque ces dernières ne sont pas requises. Ceci permet de réduire également les consommations de stockage et de bande passante. D'autre part, chaque fois que la stratégie de réplication comparée dépasse le seuil de temps de réponse, il y aura une violation SLA. En conséquence, le fournisseur devrait payer une pénalité au locataire, alors qu'il n'y aura pas de pénalité à payer lors de l'utilisation de notre stratégie proposée puisque le temps de réponse obtenu ne dépasse pas le seuil de temps de réponse. Cela explique pourquoi nous avons des réplications plus importantes avec la stratégie proposée.

Concernant la consommation des ressources par le fournisseur, en dépit de la consommations plus importante de stockage (plus de répliques créés), la réduction de la consommation de la bande passante est significative avec notre stratégie. Néanmoins, les couts monétaires de stockage sont beaucoup moins importants que ceux de l'utilisation de la bande passante. En conséquence, la stratégie déjà existante dans MongoDB consomme plus de ressources, c'est-à-dire que le profit du fournisseur est moins important.

4.7. Conclusion

Dans ce chapitre, nous avons présenté la mise en œuvre de la stratégie de réplication de données proposée dans Mongodb. Nous avons validé la stratégie proposée avec une évaluation expérimentale. Nous avons comparé la stratégie de réplication existante dans Mongodb avec notre stratégie proposée en termes de temps de réponse et de nombre de réplications déclenchées. Nous avons également évalué les ressources utilisées par les deux stratégies. Les résultats obtenus confirment que la stratégie proposée utilise moins de ressources. Elle est meilleure lorsqu'on considère à la fois la satisfaction du temps de réponse pour le locataire et la rentabilité économique pour le fournisseur.

CHAPITRE 5 : ETUDE DE CAS CONTEXTE DE PLANIFICATION DES PROJETS TERRITORIAUX

Sommaire

CHAPITRE 5 : ETUDE DE CAS CONTEXTE DE PLANIFICATION DES PROJETS TERRITORIAUX 84

5.1. Introduction 85

5.2. Etude de cas (les opérateurs téléphoniques) 86

5.7. Discussion 91

5.8. Conclusion 93

5.1. Introduction

Au cours des dernières années les organisations urbaines passent d'une activité basée sur le volume à une activité basée sur la valeur, ce qui nécessite beaucoup d'efforts de la part des administrateurs et des particuliers urbains pour être plus productifs et efficaces. Cela améliorera la pratique en milieu urbain par exemple : assurer une vie domestique et professionnelle sûre, organisée et agréable aux résidents des villes (nouvelles et établies).

Étant donné que les systèmes d'informations urbaines génèrent chaque fois des quantités énormes de documents, il semble que le monde atteint le niveau de surcharge de données. Il est maintenant évident que, pour traiter de tels volumes de données, une capacité énorme est nécessaire en termes de ressources de stockage et de calcul. Alors que la croissance de la capacité est limitée par l'évolution du matériel et des technologies, la croissance du volume de données est en fait illimitée. Dans ce chapitre nous avons validé la stratégie proposée avec un exemple de cas d'application, qui offre la possibilité de gérer les infrastructure des opérateurs téléphoniques pour obtenir de meilleures décisions dans le domaine de la planification du territoire urbain **.**

5.2. Etude de cas (les opérateurs téléphoniques)

Chaque projet urbain concerne différents domaines tels que les infrastructures (eau et gaz), la construction (maisons, écoles, hôpitaux ...), le transport, la communication (réseau social, réseau téléphonique), chacun de ces domaines a ses informations et données de sources mixtes.

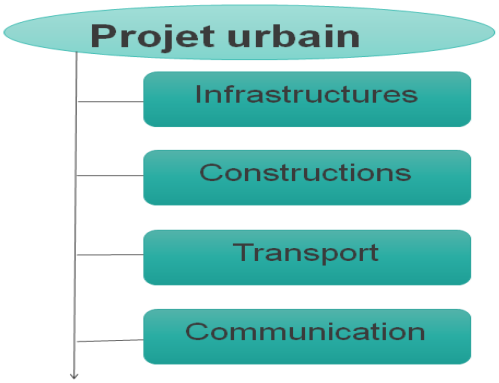


Figure 5.1. Type des projets urbain

Nous nous sommes concentrée sur le domaine de communication; Chaque opérateur téléphonique, a ces propre caractéristiques telles que les abonnées, les services, et encore un ensemble des infrastructures ex. ; locaux et antennes.

Figure 5.2. Caractéristiques d'un opérateur téléphonique

Les villes se développent très rapidement, alors que le abonnées, nous reprenons les problèmes suivantes :

- Surpeuplement des centres (agences) clients.
- Le nombre d'antennes sera insuffisant.
- Congestion du réseau téléphonique et de nombreux autres problèmes.
- ...

Cela nécessite plus d'infrastructures (locaux /antennes), qui peut considérer comme un projet de planification de territoire urbain

Dans le tableau suivant nous avons appliquer une projection de la starategie proposée de la replication de données selon cette cas d'application (étude de cas)

Cas d'application	Proposition
Personne	Requête {Q}
Abonnée	Sous - Requête { Q1, Q2, Qi}
Commune	Nœud
Local / Antennes	Données (à repliqué)
Wilaya	Datacenter

Tableau 5.1. Transformation des paramétres.

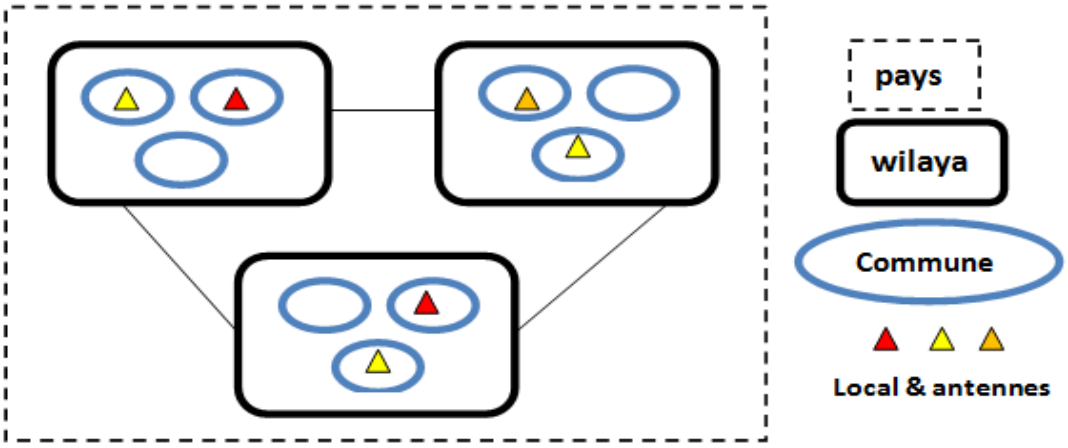


Figure 5.3. Transformation des paramétres

- **Scénario : Réseau téléphoniques**

Algorithme de décision de Réplication

Début
1. *RespQ ← temps de réponse estimé d'une requête Q.*
2. ***Si (RespQ > RespT) alors***
3. ***Sélectionner les données concernées par la réplication.***
4. Chercher le nœud approprié pour place la nouvelle réplique.
5. *P ← profit estimé par placement d'une nouvelle réplique.*
6. Si (P > 0) alors
7. Déclencher la réplication.
Fin

* L'ors d'une consultation d'un site internet (on utilisant le service 3G/4G) d'un opérateur X .

* Les utilisateurs attendent un certain temps qui n'est pas en accord avec le contrat avec l'opérateur (type service) E.g : le débits (*yy* Mo/s) .

* Donc, le responsable de l'opérateur doit sélectionner le type d'infrastructure à construire (à répliqué).

* Chercher un emplacement qui satisfait les contraintes du placement des répliques

* Tant que, le placement des nouvelle infrastructure génèrent un bénéfice ; lancer le projet de construction.

Afin de faciliter la manipulation des données dans Mongodb, nous avons utilisé l'outil '*Jetbrains webstorm'* ,qui est un IDE (Integrated Developement Envirenment) utilisé pour les langages web (HTML, CSS, JavaScript...). Après avoir installer l'outil, on doit ensuite installer les plug-in du Mongodb, qui permet l'accès aux bases des données et fournir les différentes opérations sur Mongo collections.

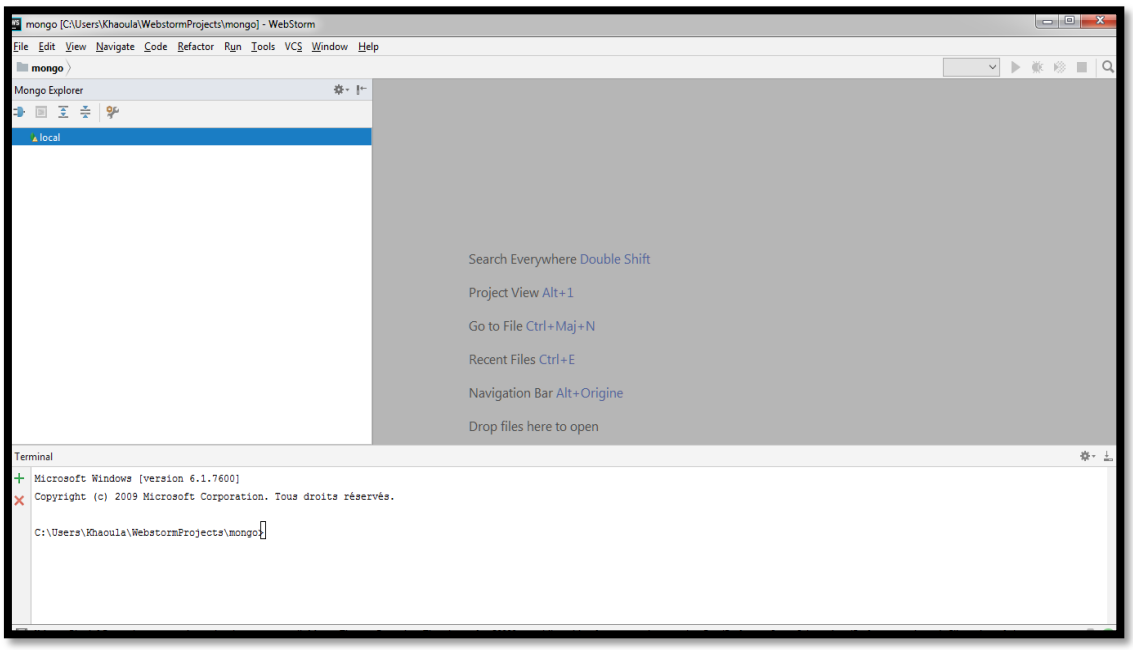
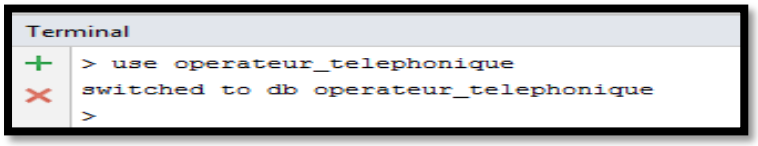


Figure 5.4: Interface graphique du 'Jetbrains webstorm'.

La première étape consiste à créer une nouvelle base de données, on utilise la commande suivante use *(operateur téléphonique)*, mais d'abord on doit se connecter au serveur Mongod, et ensuite exécuter la commande précédente (*mongod*):

```
C:\Users\Khaoula>mongod
2018-07-15T20:30:39.801+0100 I CONTROL  [initandlisten] MongoDB starting : pid=3
612 port=27017 dbpath=C:\data\db\ 64-bit host=DELL
2018-07-15T20:30:39.802+0100 I CONTROL  [initandlisten] targetMinOS: Windows Vis
ta/Windows Server 2008
2018-07-15T20:30:39.802+0100 I CONTROL  [initandlisten] db version v3.4.2
2018-07-15T20:30:39.802+0100 I CONTROL  [initandlisten] git version: 3f76e40c105
fc223b3e5aac3e20dcd026b83b38b
2018-07-15T20:30:39.802+0100 I CONTROL  [initandlisten] allocator: tcmalloc
2018-07-15T20:30:39.803+0100 I CONTROL  [initandlisten] modules: none
2018-07-15T20:30:39.803+0100 I CONTROL  [initandlisten] build environment:
2018-07-15T20:30:39.803+0100 I CONTROL  [initandlisten]     distarch: x86_64
2018-07-15T20:30:39.803+0100 I CONTROL  [initandlisten]     target_arch: x86_64
2018-07-15T20:30:39.804+0100 I CONTROL  [initandlisten] options: {}
2018-07-15T20:30:39.805+0100 I -        [initandlisten] Detected data files in C
:\data\db\ created by the 'wiredTiger' storage engine, so setting the active sto
rage engine to 'wiredTiger'.
2018-07-15T20:30:39.806+0100 I STORAGE  [initandlisten] wiredtiger_open config:
create,cache_size=3535M,session_max=20000,eviction=(threads_max=4),config_base=f
alse,statistics=(fast),log=(enabled=true,archive=true,path=journal,compressor=sn
appy),file_manager=(close_idle_time=100000),checkpoint=(wait=60,log_size=2GB),st
atistics_log=(wait=0),
2018-07-15T20:30:41.334+0100 I CONTROL  [initandlisten]
2018-07-15T20:30:41.335+0100 I CONTROL  [initandlisten] ** WARNING: Access contr
ol is not enabled for the database.
2018-07-15T20:30:41.336+0100 I CONTROL  [initandlisten] **          Read and wri
te access to data and configuration is unrestricted.
2018-07-15T20:30:41.337+0100 I CONTROL  [initandlisten]
2018-07-15T20:30:41.338+0100 I CONTROL  [initandlisten] Hotfix KB2731284 or late
r update is not installed, will zero-out data files.
2018-07-15T20:30:41.339+0100 I CONTROL  [initandlisten]
2018-07-15T20:30:41.611+0100 W FTDC     [initandlisten] Failed to initialize Per
formance Counters for FTDC: WindowsPdhError: PdhExpandCounterPathW failed with '
L?objet spécifié n?a pas été trouvé sur l?ordinateur.' for counter '\Memory\Avai
lable Bytes'
2018-07-15T20:30:41.611+0100 I FTDC     [initandlisten] Initializing full-time d
iagnostic data capture with directory 'C:/data/db/diagnostic.data'
2018-07-15T20:30:41.616+0100 I NETWORK  [thread1] waiting for connections on por
t 27017
```



Ensuite on va ajouter une collection " Abonnés", qui se compose d'un ensemble des documents sous format JSON (Java Script Object Notation), c'est pour cela qu'on va insérer (ajouter) un ensemble des documents (abonnés), qui sont caractérisés par les attributs décrits ci-dessous:

Abonné

({

"Full_name": valeur,

"Age": valeur,

"Gender": valeur,

"Id_type": valeur,

"Email": valeur,

"Phone_nbr": valeur,

"Address": valeur,

"Registration_date": valeur,

"Received_calls": valeur,

"Notification": valeur,

"Consulted_social_medias": valeur

})

Sachant que lorsqu'on ajoute un nouveau document, Mongodb génère d'une façon automatique un identifiant unique en format hexadécimal.

Pour l'insertion d'un ensemble d'objets sous format JSON, nous avons choisi un générateur gratuit de données sous format JSON (*JSON Generator*), qui permet de générer toutes les données aléatoirement, que nous voulons dans un modèle souple et puissant à enregistrer sur nos serveurs pour une utilisation ultérieure.

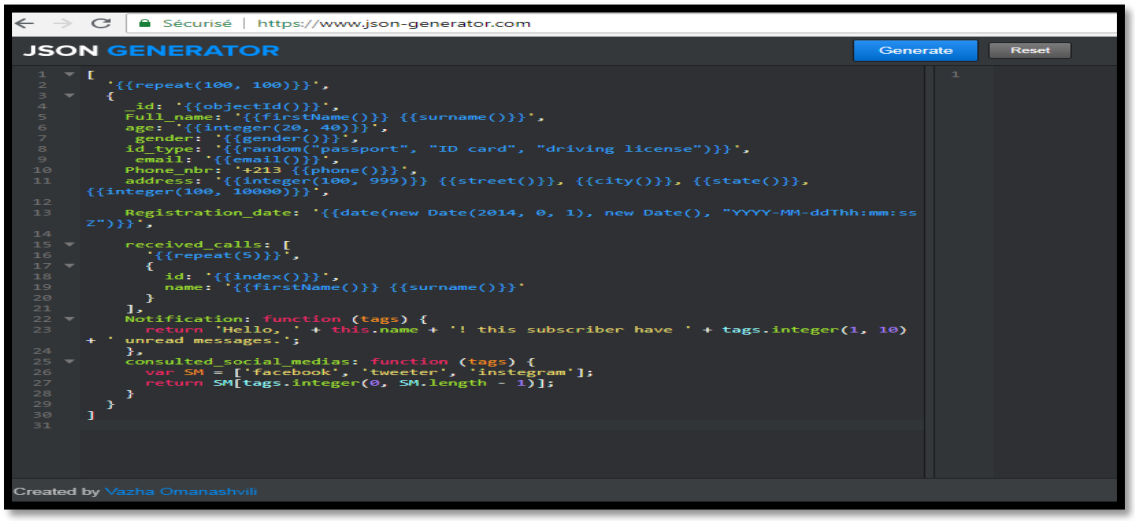


Figure 5.5: Insertion des JSON objects avec *JSON GENERATOR* tool

Après la génération de l'ensemble des documents qu'on va insérer, nous obtiendrons les documents sous format suivant :

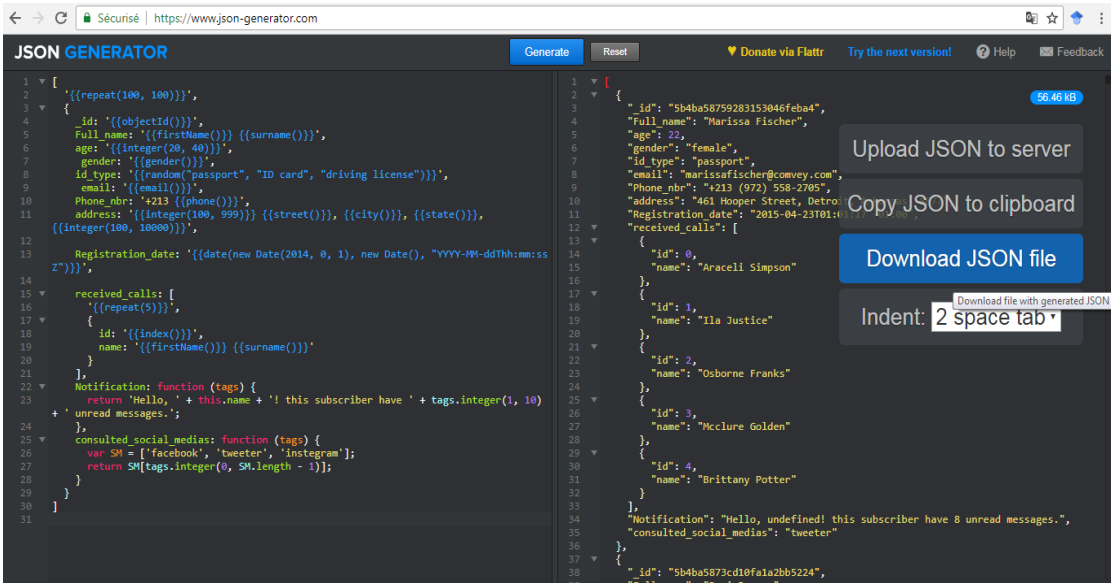


Figure 5.6 : Format du fichier *JSON* après l'opération de la génération

La prochaine étape consiste à faire des requêtes (opérations), dont le premier cas représente le test sans l'utilisation de la réplication de données, alors que le deuxième cas représente le test avec l'implémentation de la réplication de données. À la fin on va comparer les performances(temps de réponse) de chaque cas. la figure ci-dessous représente les résultats obtenus.

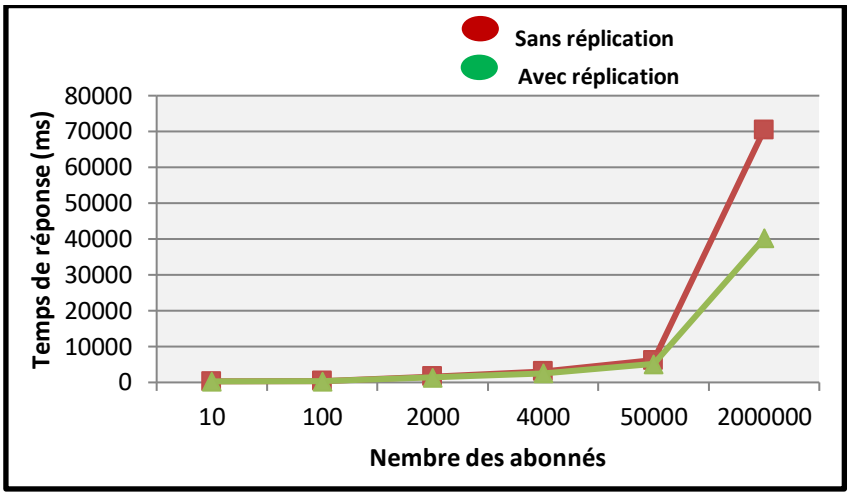


Figure 5.7 : Résultat obtenu des tests

Selon la figure 5.9 Nous avons atteint notre objectif de prouver que lorsqu'on introduit la réplication de données avec l'étude de cas que nous avons choisie, les opérateurs téléphoniques accèdent aux données plus rapidement (performance), avec une garantie sur la disponibilité des données, et même en cas de pannes (tolérance aux pannes).

MongoDB met à disposition un concept de réplication des données. Il permet de garantir la haute disponibilité des données et dans certains cas d'augmenter les capacités en lecture.

5.7. Discussion

Bien que la vérification ait prouvé son utilité et celle de l'approche proposée, il y a encore un certain nombre de limites à noter. Le plus gros problème, tout au long de ce travail,, est l'absence de contact direct avec la pratique de notre approche. Cela se voit en plusieurs points. Comme l'architecture de référence doit être basée sur les meilleures pratiques et les concepts éprouvés de la pratique, il s'agit certainement d'un problème qui ne peut être totalement compensé par la prise en compte de ses meilleures pratiques dans la littérature. Enfin, cela n'a pas permis une vérification plus rigide, basée sur des scénarios et des entrevues, de l'architecture de référence pour vérifier la pertinence ou pour appliquer l'architecture de référence dans une situation de projet concrète.

Le Big Data et le cloud computing modifient la façon dont de nombreuses applications sont développées. Les bases de données relationnelles ont dominé les industries pendant de nombreuses années, mais les bases de données NoSQL attirent maintenant l'attention des développeurs d'applications pour les raisons suivantes: (i) Les bases de données NoSQL fournissent un modèle de données flexibles et sans schéma, le plus approprié pour les grands utilisateurs et les big data. (ii) Les bases de données NoSQL ont une capacité à évoluer de façon spectaculaire pour prendre en charge les utilisateurs

mondiaux et les mégas données. (iii) La base de données NoSQL fournit une performance améliorée pour satisfaire les attentes des grands utilisateurs sans compromettre l'évolutivité.

5.8. Conclusion

La planification urbaine commence à être disponible à partir de sources non conventionnelles et a remis à l'honneur les modèles urbains intégrés. Les données basées sur la télédétection, les réseaux sociaux en ligne, la billetterie intelligente de transit, l'utilisation du téléphone mobile et les transactions par carte de crédit ont un dénominateur commun: elles contiennent toutes des informations géo localisées.

En conséquence, nous passons de données structurées, statiques, démographiques et d'activité économique à des données dynamiques non structurées capables de fournir de nouvelles perspectives sur la dynamique urbaine. Il existe de nombreuses méthodes et outils pour gérer et planifier le projet urbain. Dans ce contexte l'adoption de nouvelles technologies nécessite de traiter, de découvrir et d'analyser ces ensembles de données massives qui ne peuvent pas être traitées à l'aide de bases de données et d'architectures traditionnelles en raison du manque de ressources en termes de calcul et de stockage. L'analyse haute performance représente l'une des approches innovantes qui peuvent être appliquées sur les volumes croissants, la vitesse et la variété des données.

CONCLUSION GENERALE

Sommaire

CONCLUSION GENERALE 94

1. Contribution 95

2. Perspectives 97

1. Contribution

La plupart des études dans la littérature traitant des stratégies de réplication dans MongoDB concentrent leur attention sur l'évaluation de la réplication et la comparaison de différentes stratégies de réplication de données. Certains travaux ont comparé les performances de la stratégie de réplication existante dans Mongodb avec d'autres stratégies de réplication établies pour les systèmes NoSQL ainsi que pour les systèmes de gestion de bases de données relationnelles. Cependant, aucun d'entre eux n'attire l'attention sur la gestion des répliques dans Mongodb lorsqu'il est utilisé comme back-end par un fournisseur de services cloud. De plus, ils négligent le profit du fournisseur.

Dans cette thèse, nous avons proposé une nouvelle stratégie de réplication de données pour un système Nosql orienté documents et plus précisement, le système MongoDB. Cette stratégie vise à satisfaire simultanément les exigences du locataire ainsi que le profit économique du fournisseur. Un certain nombre de contributions ont été faites pendant la durée de cette thèse. Ces contributions peuvent être résumées comme suit:

(i) Une nouvelle classification des stratégies de réplication de données dans les systèmes cloud. Nous avons classé les stratégies de réplication en fonction de quatre critères: (a) nature de la réplication de données *(statique vs. dynamique)*, (b) équilibrage de la charge de travail *(réactif vs. proactif)*, (c) bénéficiaire *(fournisseur vs. client)*, et (d) fonction objective visée. La taxonomie que nous avons proposée peut être un guide utile pour les IT managers pour sélectionner la stratégie de réplication de données qui convient le mieux à leur organisation.

(ii) Une nouvelle stratégie de réplication de données a été proposé. Elle vise à satisfaire à la fois les performances pour le locataire et la rentabilité du fournisseur. La satisfaction simultanée des deux critères, à savoir la satisfaction du temps de réponse et la rentabilité du fournisseur, est recherchée pour chaque exécution d'une requête utilisateur. Lorsqu'une requête est soumise pour exécution, la stratégie de réplication de données proposée identifie si la réplication de données est nécessaire. En fait, la réplication n'est envisagée que si le temps de réponse estimé dépasse le temps de réponse seuil établi dans le contrat SLA. De plus, la réplication doit être profitable pour le fournisseur en considérant cette réplication. Par la suite, une création de réplique est faite de telle sorte qu'elle satisfasse les performances requises par l'utilisateur et le profit du fournisseur. Nous nous sommes basés à la fois sur la popularité des données et sur la charge de travail des nœuds pour identifier les données concernées par la réplication. En ce qui concerne le placement d'une nouvelle réplique, un algorithme de placement de répliques a été proposé de telle sorte à tirer profit de la localité du point de vue bande passante. L'ajustement du nombre de répliques se fait également de manière dynamique afin de permettre une gestion souple des ressources afin de réduire la consommation de ces ressources. Le nombre de répliques est ajusté en fonction de la valeur du temps de réponse estimé d'une requête. Lorsque cette valeur est

significativement plus petite que le seuil de réponse, nous diminuons ce facteur de réplique tel que le fournisseur augmente son bénéfice.

Nous avons validé la stratégie proposée à travers une série de simulation. La stratégie proposée a été comparée à la stratégie de réplication dèja existante dans Mongodb. Les différentes exepériences ont montré que notre stratégies satisfait non seulement les exigences de performances du locataire (temps de réponse) mais également prend en compte le profit du fournisseur, négligé dans la stratégie existante dans Mongodb. Notre stratégie permet également la consommation de moins de ressources suite à la gestion élastique des répliques ce qui diminue les dépenses du fournisseur. En conséquence, le profit du fournisseur sont plus grands.

2. Perspectives

Certaines nouvelles orientations de recherche ont émergé. Nous discutons ci-dessous de certains de ces travaux futurs dans notre domaine de recherche.

(i) Nous avons l'intention d'étendre le processus de décision de création de répliques en acceptant de perdre un certain profit lorsque le fournisseur exécute les requêtes d'un locataire important dans un environnement multi-locataires. Cela devrait être avantageux pour le fournisseur sur une longue période à la condition que le fournisseur fasse de nouveau un profit avec le même locataire par la suite.

(ii) Nous envisageons la proposition d'une gestion des pénalités. Si le fournisseur parvient à satisfaire le SLA, aucune pénalité n'est encourue pour le fournisseur. Autrement, un montant de pénalité est payé par le fournisseur au locataire. Le montant de la pénalité dépend du degré de violation du SLA. Dans la stratégie proposée, nous assurons la satisfaction du SLA en estimant le temps de réponse des requêtes. Une autre voie possible pour atteindre ce résultat est d'estimer le montant de la pénalité due à l'exécution d'une requête. De cette façon, la minimisation des pénalités deviendrait un objectif de la stratégie de réplication de données, au lieu d'une conséquence. Bien sûr, la minimisation des pénalités et la satisfaction des exigences de l'utilisateur doivent être assurés et la rentabilité a postériori reste l'objectif final à atteindre.

(iii) La stratégie proposée de la réplication de données prend les décisions de réplication avant l'exécution de chaque requête utilisateur. Un avantage significatif de cette approche est de répondre immédiatement à toute évolution de la charge et de la popularité des requêtes. Cependant, cela peut engendrer une charge supplémentaire due à la réplication répétitive de données. L'alternative consiste à effectuer une réplication périodique ou par groupes de requetes au lieu de l'examiner à chaque requête. Dans la réplication périodique, les données historiques pour lesquelles les requêtes sont populaires et les fragments auxquels on accède plus fréquemment pourront etre collectes. Ensuite, ces informations peuvent être utilisées à intervalles réguliers pour effectuer une optimisation globale du processus de la réplication de données. Cette approche peut également être moins coûteuse en termes de coûts indirects. Une bonne opportunité de recherche est disponible ici pour comparer les deux approches et étudier laquelle est la plus appropriée pour différents scénarios.

(iv) Nous prévoyons de mettre en œuvre la stratégie proposée dans un environnement cloud réel.

Publications et Communications

Les publications et les communications qui ont été produites tous au long de cette thèse sont les suivantes :

Publications Internationales

1. *TABET, Khaoula*, MOKADEM, Riad, LAOUAR, Mohamed Ridda. A Data Replication Strategy for Document Oriented NoSQL Databases. International Journal of Grid and Utility Computing (IJGUC), 2018. (in print)

2. *TABET, Khaoula*, MOKADEM, Riad, LAOUAR, Mohamed Ridda, Sean B. EOM. Data replication in cloud systems: a survey. *International Journal of Information Systems and Social Change (IJISSC)*, 2017, vol. 8, no 3, p. 17-33.

Communications Internationales

1. TABET, Khaoula, MOKADEM, Riad, LAOUAR, Mohamed Ridda. A New Data Replication in Cloud Systems for Performance and Profit Garantee. International Conference on Intelligent Computing and Decision Systems (ICICDS'2016), Hammamet, Tunisie. 26-28 December 2016.

2. TABET, Khaoula, MOKADEM, Riad, LAOUAR, Mohamed Ridda. Towards New Cloud-based Replication Strategy for Document Oriented NoSQL Database. 6th International Conference on Software Engineering and New Technologies (ICSENT'2017), Hammamet, Tunisie. 28-30 December 2017.

3. TABET, Khaoula, MOKADEM, Riad, LAOUAR, Mohamed Ridda. Towards a New Data Replication Strategy in MongoDB systems. 4th ACM - International Conference of Computing for Engineering and Sciences Technology (ICCES' 2018), Kuala Lumpur, Malaysia. 14-16 July 2018.

4. BRADJI, Louardi; TABET, Khaoula; LAOUAR, Mohamed Ridda. A New Approach to Big Data based Urban Territory Planning. EWG-DSS 2nd International Conference on Decision Support System Technology (ICDSST 2016), Plymouth , UK

Communications nationales

1. TABET, Khaoula; LAOUAR, Mohamed Ridda. Approach based Big Data for Urban Territory Planning. JOMI 2015 (Journées Ouvertes sur les Mathématiques et l'Informatique) Tébessa, Algérie.14-15 Mai 2015.

Workshops

1. TABET, Khaoula. Getting started with big data tools. Doctoral students in computer science : workshops and practices, April 15- 19, 2018.

2. TABET, Khaoula. How computer science is revolutionizing medicine? a Cambridge university experience. Doctoral students in computer science : workshops and practices, Tebessa, April 15- 19, 2018.

3. TABET, Khaoula. Deep learning : principles, techniques and tools. Doctoral students in computer science : workshops and practices, Tebessa, April 15- 19, 2018.

4. TABET, Khaoula. Scientific paper : how to succeed in redaction and publication? . Doctoral students in computer science : workshops and practices, Tebessa, April 15- 19, 2018

5. TABET, Khaoula. The Islamic golden age and its impact in science and technologies. Doctoral students in computer science : workshops and practices, Tebessa, April 15- 19, 2018

6. TABET, Khaoula. The Smart -environments; from theory to practice. Doctoral students in computer science : workshops and practices, Tebessa, April 15- 19, 2018

Bibliographie

[1] Eddy Meylan, « Bases de données : Réplication des données » Support de cours. Haute Ecole spécialisée de Suisse Occidentale, Informatique de Gestion, (Février 2005).

[2] Matthieu Exbrayat, « Bases de Données Réparties : Concepts et Techniques », notes de cours. ULP Strasbourg. (Décembre 2007).

[3] Mokadem, R., Hameurlain, A. Data replication strategies with performance objective in data grid systems: a survey. Int. J. of Grid and Utility Computing 6(1), pp. 30-46, (2015)

[4] Eric, A. and Lozano, M.B. (2010) 'Executive's Guide to cloud computing', John Wiley & Sons, ISBN-13: 978-0470521724

[5] Vaquero, L. M., Rodero-Merino, L., Caceres, Juan. and Lindner, M. (2009) 'A Break in the Clouds: Towards a Cloud Definition', ACM SIGCOMM Computer Communication Review, Vol. 39, No. 1, pp. 50-55.

[6] Mell, P. (2011) 'The NIST Definition of Cloud Computing' Timothy Grance Special Publication 800-145, National Institute of Standards and Technology, Gaithersburg.

[7] Yuanshun, D, Yanping, X. and Gewei, Z (2009) 'Self-healing and Hybrid Diagnosis in Cloud Computing', Proceedings CloudCom of 1st International Conference on Cloud Computing, Beijing, China, pp. 45-56.

[8] Http://whatiscloud.com/cloud_characteristics/multi_tenancy. Consulter le 16/04/2018.

[9] Http://www.expertglossary.com/cloud-computing/definition/linearly-scalable. Consulter le 16/04/2018

[10] Kung-Kiu, L., Winfried, L. and Ernesto, P. (2013) 'Service-Oriented and cloud computing', Lecture Notes in Computer Science, Springer Berlin Heidelberg, ISBN: 978-3-642-40651-5.

[11] Buyya, R., Garg, S.K. and Calheiros, R.N. (2011) 'SLA-oriented resource provisioning for cloud computing: Challenges, architecture, and solutions', Proceedings of the International Conference on Cloud and Service Computing (CSC), Hong Kong, China, pp. 1-10.

[12] Www.broadcom.com/collateral/wp/Virtualization-WP100-R.pdf . Consulter le 13/03/2018

[13] Http://www.hosting.com/resources/white-papers/what-cloud-computing-means-to-you-efficiency-flexibility/success-cloud-computing-means/ .20/04/2018

[14] Http://www.bibsonomy.org/bibtex/2a5228c1a4057026333619d53f753e449/vonposer . Consulter le 13/03/2018

[15] Https://cloud.google.com/developers/articles/building-high-availability-applications-on-google-compute-engine/. Consulter le 13/03/2018

[16] Http://en.wikipedia.org/wiki/EMC_Corporation .Consulter le 14/02/2018.

[17] Wang, G., & Tang, J. (2012, August). The nosql principles and basic application of cassandra model. In Computer Science & Service System (CSSS), 2012 International Conference on (pp. 1332-1335). IEEE..),

[18] Hofmann, P. and Woods, D. (2010) 'cloud computing: The Limits of Public Clouds for Business Applications' IEEE Internet Computing, Vol. 14, No. 6, pp. 90–93.

[19] Milojicic, D. and Wolski, R. (2011) 'Eucalyptus: Delivering a Private Cloud. Computer, Vol. 44, No. 4, pp. 102–104.

[20] Http://www.strategicitarchitecture.com/2009/11/flight-plan-deploying-the cloud/. Consulter le 14/05/2018

[21] Matteo Di Maglie, Adoption d'une solution NoSQL dans l'entreprise, Travail de Bachelor réalisé en vue de l'obtention du Bachelor HES, Carouge, 12 septembre 2012 Haute École de Gestion de Genève (HEG-GE).

[22] Kouedi Emmanuel, Approche de migration d'une base de données relationnelle vers une base de données NoSQL orientée colonne, Mémoire présenté en vue de l'obtention du diplôme de MASTER II RECHERCHE, Option : S.I & G.L ; Université de YAOUNDE I. Mai 2012.

[23] Adriano Girolamo PIAZZA, NoSQL Etat de l'art et benchmark ;Travail de Bachelor réalisé en vue de l'obtention du Bachelor HES ; Genève, 9 octobre 2013 Haute École de Gestion de Genève (HEG-GE).

[24] Sadalage, P. (2014). NoSQL databases: an overview. Published October, 1.

[25] NoSQL Databases and cloud computing, Written By Michael Hayes , On May 30, 2013 .

[26] Pérez, J. M., García-Carballeira, F., Carretero, J., Calderón, A., & Fernández, J. (2010). Branch replication scheme: A new model for data replication in large scale data grids. *Future Generation Computer Systems*, 26(1), 12-20.

[27] Loukopoulos, T., Lampsas, P., & Ahmad, I. (2005). Continuous replica placement schemes in distributed systems. *Proceedings of the 19th annual international conference on Supercomputing*: ACM. 284-292.

[28] Benoit, A., Rehn-Sonigo, V., & Robert, Y. (2008). Replica placement and access policies in tree networks. *IEEE Transactions on Parallel and Distributed Systems*, 19(12), 1614-1627.

[29] Xhafa, F., Kolici, V., Potlog, A.-D., Spaho, E., Barolli, L., & Takizawa, M. (2012). Data replication in P2P collaborative systems. *The 7th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*. Victoria, BC: IEEE. 49–57 .

[30] Mansouri, Y., Azad, S. T., & Chamkori, A. (2014). Minimizing cost of k -replica in hierarchical data grid environment. *2014 IEEE 28th International Conference on Advanced Information Networking and Applications*: IEEE. 1073-1080.

[31] Tos, U., Mokadem, R., Hameurlain, A., Ayav, T., & Bora, S. (2015). Dynamic replication strategies in data grid systems: A survey. *The Journal of Supercomputing*, 71(11), 4116–4140.

[32] Wei, Q., Veeravalli, B., Gong, B., Zeng, L., & Feng, D. (2010): CDRM A cost-effective dynamic replication management scheme for cloud storage cluster. *2010 IEEE International Conference on Cluster Computing*: IEEE. 188-196.

[33] Chervenak, A., Deelman, E., Foster, I., Guy, L., Hoschek, W., Iamnitchi, A., Kesselman, C., Kunszt, P., Ripeanu, M., Schwartzkopf, B., Stockinger, H., Stockinger, K., & Tierney, B. (2002). Giggle: A framework for constructing scalable replica location services. *Proceedings of the 2002 ACM/IEEE Conference on Supercomputing* IEEE Computer Society Press. 1-17.

[34] Čibej, U., Slivnik, B., & Robič, B. (2005). The complexity of static data replication in data grids. *Parallel Computing* 31 (8-9), 900-912.

[35] Sashi, K., & Thanamani, A. S. (2011). Dynamic replication in a data grid using a modified BHR region based algorithm. *Future Generation Computer Systems*, 27(2), 202–210.

[36] Amjad, T., Sher, M., & Daud, A. (2012). A survey of dynamic replication strategies for improving data availability in data grids. *Future Generation Computer Systems*, 28(2), 337-349.

[37] Grace, R. K., & Manimegalai, R. (2014). Dynamic replica placement and selection strategies in data grids -- a comprehensive survey. *Journal of Parallel and Distributed Computing*, 74(2), 2099-2108.

[38] Doğan, A. (2009). A study on performance of dynamic file replication algorithms for real-time file access in data grids. *Future Generation Computer System* 25(8), 829–839

[39] Steen, M. v., & Pierre, G. (2010). Replicating for performance: Case studies. In B. Charron-Bost, F. Pedone, & A. Schiper (Eds.), *Replication, theory and practice*. Berlin: Springer, 73–89.

[40] Tabet, K., Mokadem, R., Laouar, M. R., & Eom, S. (2017). Data replication in cloud systems: a survey. *International Journal of Information Systems and Social Change (IJISSC)*, *8*(3), 17-33.

[41] Ghemawat, S., Gobioff, H., & Leung, S.-T. (2003). The google file system. *ACM SIGOPS Operating System Review* 37(5), 29-43.

[42] Bai, X., Jin, H., Liao, X., Shi, X., & Shao, Z. (2013). RTRM: A response time-based replica management strategy for cloud storage system. In J. J. J. H. Park, H. R. Arabnia, C. Kim, W. Shi, & J.-M. Gil (Eds.), *Grid and pervasive computing: 8th international conference, GPC (grid and pervasive computing) 2013 and colocated workshops, seoul, korea, may 9-11, 2013.*

[43] Silvestre, G. D. S., Monnet, S., Krishnaswamy, R., & Sens, P. (2012). AREN: A popularity aware replication scheme for cloud storage. *IEEE 18th International Conference on Parallel and Distributed Systems (ICPADS 2012)* Singapore. 189-196.

[44] Hussein, M.-K., & Mousa, M.-H. (2014). A light-weight data replication for cloud data centers environment. *International Journal of Engineering and Innovative Technology (IJEIT)*, 1(6), 169-17.

[45] Sakr, S., & Liu, A. (2012). Sla-based and consumer-centric dynamic provisioning for cloud databases. *2012 IEEE 5th International Conference on cloud computing (CLOUD)*: IEEE. 360-367.

[46] Sousa, F. R. C., & Machado, J. C. (2012). Towards elastic multi-tenant database replication with quality of service. *Proceedings of 2012 IEEE/ACM Fifth International Conference on Utility and cloud computing* IEEE. 168-175

[47] Bonvin, N., Papaioannou, T. G., & Aberer, K. (2011). Autonomic sla-driven provisioning for cloud applications. *Proceedings of the 2011 11th IEEE/ACM international symposium on cluster, cloud and grid computing*: IEEE Computer Society. 434-443.

[48] Kirubakaran, S., Valarmathy, S., & Kamalanathan, C. (2013). Data replication using modified D2RS in cloud computing for performance improvement. *Journal of Theoretical and Applied Information Technology* 58(2), 460-470.

[49] Tos, U., Mokadem, R., Hameurlain, A., Ayav, T., & Bora, S. (2016). A performance and profit oriented data replication strategy for cloud systems. *The IEEE International Conference on Cloud and Bigdata Computing* Toulouse.

[50] Xue, M., Shen, J., & Guo, X. (2015). Replica placement in cloud storage based on minimal blocking probability. *CENet2015* Shanghai, China.

[51] Lin, J.-W., Chen, C.-H., & Chang, J. M. (2013). QoS-aware data replication for data-intensive applications in cloud systems. *IEEE Transactions on Cloud Computer Networks*, 1(1), 101–115.

[52] Lamehamedi, H., Shentu, Z., Szymanski, B., & Deelman, E. (2003). Simulation of dynamic data replication strategies in data grids. *IPDPS '03 Proceedings of the 17th International Symposium on Parallel and Distributed Processing*. 2003: IEEE Computer Society. Page 100.2

[53] Chang, R. S., Huang, N. Y., & Chang, J. S. (2006). A predictive algorithm for replication optimization in data grid. *Proceedings of ICS 2006*. Taiyuan, Taiwan. 199–204.

[54] Tang, M., Lee, B.-S., Yeo, C.-K., & Tang, X. (2005). Dynamic replication algorithms for the multi-tier data grid. *Future Generation Computer Systems* 21(5), 775–790.

[55] Lei, M., & Vrbsky, S. V. (2006). A data replication strategy to increase data availability in data grids. *Proceedings of the 2006 International Conference on Grid Computing & Applications, GCA 2006*. Las Vegas, NV. 221–227.

[56] Ranganathan, K., & Foster, I. (2001). Identifying dynamic replication strategies for a high performance data grid. *Grid Computing — GRID 2001: Second International Workshop Denver, CO, USA, Proceedings*: Springer Berlin Heidelberg. pp 75-86.

[57] Lei, M., Vrbsky, S. V., & Hong, X. (2008). An on-line replication strategy to increase availability in data grids. *Future Generation Computer Systems*, 24 (2), 85–98.

[58] Park, S.-M., Kim, J.-H., Ko, Y.-B., & Yoon, W.-S. (2004). Dynamic data grid replication strategy based on internet hierarchy. *Grid and cooperative computing: Second international workshop, gcc 2003, shanghai, china, december 7-10, 2003, revised papers, part ii*. Berlin Heidelberg: Springer Berlin Heidelberg, 838-846.

[59] Bell, W. H., Cameron, D. G., Carvajal-Schiaffino, R., Millar, A. P., Stockinger, K., & Zini, F. (2003). Evaluation of an economy-based file replication strategy for a data grid. *Proceedings of the 3rd IEEE/ACM international symposium on cluster computing and the grid* IEEE, 661-668.

[60] Milani, B. A., & Navimipour, N. J. (2016). A comprehensive review of the data replication techniques in the cloud environments: Major trends and future directions. *Journal of Network and Computer Applications*, 64, 229–238.

[61] Long, S.-Q., Zhao, Y.-L., & Chen, W. (2014a). MORM: A multi-objective optimized replication management strategy for cloud storage cluster .*Journal of Systems Architecture*, 60(2), 234–244.

[62] Chang, R. S., Huang, N. Y., & Chang, J. S. (2006). A predictive algorithm for replication optimization in data grid. *Proceedings of ICS 2006*. Taiyuan, Taiwan. 199–204.

[63] Li, W., Yang, Y., Chen, J., & Yuan, D. (2012). A cost-effective mechanism for cloud data reliability management based on proactive replica checking. *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*: IEEE. 564-571.

[64] Jennings, B., & Stadler, R. (2015). Resource management in clouds: Survey and research challenges. *Journal of Network and Systems Management*, 23(3), 567–619.

[65] Wang, Z., Li, T., Xiong, N., & Pan, Y. (2012). A novel dynamic network data replication scheme based on historical access record and proactive deletion. *The Journal of Supercomputing*, 62, 227-250.

[66] Lee, H.-J., Kim, M.-S., Hong, J. W., & Lee, G.-H. (2002). QoS parameters to network performance metrics mapping for sla monitoring. *KNOM Review*, 5(2).

[67] Long, S.-Q., Zhao, Y.-L., & Chen, W. (2014b). MORM: A multi-objective optimized replication management strategy for cloud storage cluster. *Journal of Systems Architecture*, 60(2), 234–244.

[68] Cecchet, E., Singh, R., Sharma, U., & Shenoy, P. (2011). Dolly: Virtualization-driven database provisioning for the cloud. *VEE'11: Proceedings of the 2011 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*. Newport Beach, California, USA. 51-62.

[69] Sharma, U., Shenoy, P., Sahu, S., & Shaikh, A. (2011). A cost-aware elasticity provisioning system for the cloud. *Proceedings of 2011 31st International Conference on Distributed Computing Systems (ICDCS)*: IEEE. 559–570.

[70] Ye, Z., Li, S., & Zhou, J. (2014). A two-layer geo-cloud based dynamic replica creation strategy. *Applied Mathematics & Information Sciences: The International Journal Applied Mathematics & Information Sciences* 8(1), 431-440.

[71] Gill, N. K., & Singh, S. (2014). Dynamic cost-aware re-replication and rebalancing strategy in cloud system. In S. C. Satapathy, B. N. Biswal, S. K. Udgata, & J. K. Mandal (Eds.), *Proceedings of the 3rd international conference on frontiers of intelligent computing: Theory and applications (ficta) 2014*: Springer International Publishing, 39-47.

[72] Curino, C., Jones, E. P. C., Madden, S., & Balakrishnan, H. (2011). Workload-aware database monitoring and consolidation. *SIGMOD '11 Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. Athens, Greece: ACM. 313-324.

[73] Trummer, I., & Koch, C. (2014). Approximation schemes for many-objective query optimization. *Proceedings of the 2014 ACM SIGMOD international conference on Management of data,*. Snowbird, Utah, USA 1299-1310.

[74] M. Van Steen, Pierre G.. Replicating for performance: case studies. In: Charron2Bost B, Pedone F, Schiper A (eds) Replication, Theory and Practice. LNCS,vol 5959. Springer, Berlin, pp 73–89. Chapter 5, (2010)

[75] N. Mansouri, G.H. Dastghaibyfard. A dynamic replica management strategy in Data Grid. Journal of Network and Computer Applications, 35 (4), pp. 1297–1303, (2012).

[76] R. M. Rahman, K. Barker, R. Alhajj, Replica placement strategies in data grid, Journal of Grid Computing 6 (1), pp. 103–123, (2008).

[77] Y. F. Lin, P. Liu, J.J. Wu, Optimal placement of replicas with locality assurance. Proc. Of the Int. Conf. on Parallel and Distributed Computing, (2006).

[78] W. Zhao, X. Xu, Z. Wang, Y. Zhang, S. He. A Dynamic Optimal Replication Strategy in Data Grid Environment. Proc. of Int. Conf. on Internet Technology and Applications, pp. 124, (2010).

[79] Y. F. Lin, J.J. Wu, P. Liu, A list2based strategy for optimal replica placement in data grid systems, in: 37th international Conference on Parallel Processing, pp. 198–205, (2008).

[80] Eom, S. B. (2016). Longitudinal author cocitation mapping: The changing structure of decision support systems research (1969-2012). Hanover, MA: Now Publishers (May 25, 2016).

[81] Imene Benatia., Mohamed Ridda Laouar., Hakim Bendjenna., & Sean B. Eom (2016): Implementing a Cloud-Based Decision Support System in a Private Cloud: The Infrastructure and the Deployment Process. The International Journal of Decision Support System Technology IJDSST 8(1): 25-42.

[82] Imene Benatia, Mohamed Ridda Laouar, Sean B. Eom, Hakim Bendjenna: Incorporating the Negotiation Process in Urban Planning DSS. The *International Journal of Information Systems in the Service Sector* IJISSS 8(2): 14-29 (2016).

[83] Jennings, B., & Stadler, R. (2015). Resource management in clouds: Survey and research challenges. *Journal of Network and Systems Management*, 23(3), 567–619.

[84] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Katz, A. Konwinski, G. Lee, D Patterson, A. Rabkin, L. Stoica, M. Zaharia, A view of cloud computing Commun. Communication of the ACM Vol. 53 (4), pp. 50–58, (2010).

[85] Stantchev, V., & Schröpfer, C. (2009). Negotiating and enforcing qos and slas in grid and cloud computing. *Advances in grid and pervasive computing*, 25-35.

[86] Chodorow, K. (2013). *MongoDB: The Definitive Guide: Powerful and Scalable Data Storage*. " O'Reilly Media, Inc.".

[87] Liu, Y., Wang, Y., & Jin, Y. (2012, July). Research on the improvement of MongoDB. Auto-Sharding in cloud environment. In *Computer Science & Education (ICCSE), 2012 7th International Conference on* (pp. 851-854). IEEE.

[88] Gu, Y., Wang, X., Shen, S., Ji, S., & Wang, J. (2015, June). Analysis of data replication mechanism in NoSQL database MongoDB. In *Consumer Electronics-Taiwan (ICCE-TW), 2015 IEEE International Conference on* (pp. 66-67). IEEE.

[89] Mohamed, H. H. H. (2015). *A new auditing mechanism for open source NoSQL database a case study on open source MongoDB database* (Doctoral dissertation, University Utara Malaysia).

[90] Lima, I., Oliveira, M., Kieckbusch, D., Holanda, M., Walter, M. E. M., Araújo, A., & Lifschitz, S. (2016, December). An evaluation of data replication for bioinformatics workflows on NoSQL systems. In *Bioinformatics and Biomedicine (BIBM), 2016 IEEE International Conference on* (pp. 896-901). IEEE.

[91] Haughian, G., Osman, R., & Knottenbelt, W. J. (2016, September). Benchmarking replication in cassandra and mongodb nosql datastores. In *International Conference on Database and Expert Systems Applications* (pp. 152-166). Springer International Publishing.

[92] Goel, S., & Buyya, R. (2007). Data replication strategies in wide-area distributed systems. In *Enterprise service computing: from concept to deployment* (pp. 211-241). IGI Global

[93] Tauro, C. J., Patil, B. R., & Prashanth, K. R. (2013). A comparative analysis of different nosql databases on data model, query model and replication model. In *Proceedings of the International Conference on ERCICA*.

[94] Membrey, P., Plugge, E., & Hawkins, T. (2010). *The definitive guide to MongoDB: the noSQL database for cloud and desktop computing*. Springer.

[95] Chodorow, K., & Dirolf, M. (2010). MongoDB: The Definitive Guide O'Reilly Media.

[96] Y. Kouki, T. Ledoux, and R. Sharrock, "Cross-layer SLA selection for cloud services," in 1st Int. Symp. Network cloud computing and Applications. IEEE, Nov. 2011, pp. 143–147

[97] Özsu, M. T., & Valduriez, P. (2011). *Principles of distributed database systems*. Springer Science & Business Media..

[98] Tos, U. (2017). *Réplication de données dans les systèmes de gestion de données à grande échelle* (Doctoral dissertation, Université de Toulouse, Université Toulouse III-Paul Sabatier).

[99] Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R., & Sears, R. (2010, June). Benchmarking cloud serving systems with YCSB. In *Proceedings of the 1st ACM symposium on* cloud computing (pp. 143-154). ACM.