TABLE DES MATIÈRES

Introduction **2**

I Gestion de la réplication de données **9**

1 Etat de l'art sur les stratégies de réplication de données **10**

1.1 Introduction 10

1.2 Stratégies de réplication dans les grilles 10

1.2.1 Stratégie de réplication statique vs dynamique 11

1.2.2 Stratégies de réplication centralisé vs décentralisé 11

1.2.3 Stratégies de réplication basées sur des fonctions objectives . . 12

1.2.4 Stratégies de réplication dynamique selon l'architecture 19

1.3 Stratégies de réplication dans le Cloud Computing 25

1.3.1 Statique vs dynamique 25

1.3.2 Stratégies de réplication de données selon les fonctions objectives 25

1.4 Conclusion 38

2 Stratégie proposée pour gérer la réplication de données 39

2.1 Introduction 39

2.2 Topologie de Cloud 40

2.3 Schéma de réplication 41

2.3.1 Phase de surveillance 42

2.3.2 Phase de traitement 43

2.3.3 Phase de déclenchement de réplication 43

2.4 Stratégie de réplication proposée 45

2.4.1 Quand et quoi répliquer ? 46

2.4.2 Estimation du bénéfice du fournisseur 46

2.4.3 Combien de répliques ? 48

2.4.4 Ordonnancement d'accès 50

2.4.5 Placement des répliques 50

2.4.6 Suppression des répliques 53

2.5 Conclusion 53

3 Evaluation expérimentale de la stratégie DRAPP 55

3.1 Introduction 55

3.2 CloudSim 56

3.3 Résultats expérimentaux 56

3.3.1 Impact du budget 57

3.3.2 Impact du nombre de centres de données 60

3.3.3 Impact du nombre de requêtes 62
3.4 Conclusion 64

II Gestion de la cohérence de données 66

4 Etat de l'art sur les stratégies de cohérence de données 67
4.1 Introduction 67
4.2 Stratégies de cohérence dans les grilles 68
4.3 Stratégies de cohérence dans les Clouds 69
4.3.1 Stratégies de cohérence faible 69
4.3.2 Stratégie de cohérence adaptative 71
4.4 Conclusion 76

5 Stratégie proposée pour gérer la cohérence de données 78
5.1 Introduction 78
5.2 Topologie de Cloud 79
5.3 Service de cohérence locale 80
5.3.1 Agent de surveillance 80
5.3.2 Agent de cohérence 81
5.3.3 Agent de gestion des conflits 85
5.4 Service de cohérence globale 92
5.5 Conclusion 92

6 Evaluation expérimentale de la stratégie AC2R 94
6.1 Introduction 94

6.2 Métriques utilisées 94
6.3 Simulation et résultats 96
6.3.1 Expérience 1 : Impact du nombre de Cloudlets 96
6.3.2 Expérience 2 : Impact du type de Cloudlet 99
6.4 Conclusion 103

Conclusion générale **104**

References **108**

TABLE DES FIGURES

2.1 Topologie de Cloud avec les régions et les sous-régions. 40

2.2 Architecture de la stratégie DRAPP 42

2.3 Phases opérationnelles 43

2.4 Principales étapes de DRAPP 44

3.1 Architecture du CloudSim [45] 57

3.2 Impact de la valeur des budgets sur le nombre de répliques 58

3.3 Impact de la valeur des budgets sur la disponibilité 59

3.4 Impact de la valeur des budgets sur le temps de réponse 59

3.5 Impact du nombre de centres de données sur l'équilibrage de charge 60

3.6 Impact du nombre de centres de données sur le temps de réponse . . 61

3.7 Impact du nombre de centres de données sur la consommation de bande passante 62

3.8 Impact du nombre de cloudlets sur le temps de réponse 63

3.9 Impact du nombre de cloudlets sur la violation du SLA 64

5.1 Modèle hiérarchique pour la gestion de la cohérence des répliques. . 79

5.2 Architecture du service de cohérence locale 80

5.3 Niveaux de cohérence du centre de données 81

5.4 Traitement d'une requête de lecture 83

5.5 Traitement d'une requête d'écriture 84

6.1 Variation du temps de réponse moyen par rapport au nombre de cloudlets 96

6.2 Variation du nombre de conflits par rapport au nombre de cloudlets 97

6.3 Variation du nombre de propagations de mise à jour par rapport au nombre de cloudlets 98

6.4 Effet du nombre de cloudlets sur la métrique β 99

6.5 Variation du temps de réponse moyen par rapport au type de cloudlets 100

6.6 Variation du nombre de conflits par rapport au type de cloudlets . . 101

6.7 Variation du nombre de messages par rapport au type de cloudlets . 102

6.8 Effet du nombre de cloudlets sur la métrique β 102

LIST OF ALGORITHMS

1 Déterminer le nombre minimum de répliques 49

2 Ordonnancement d'accès à la réplique 51

3 Optimisation du placement des nouvelles répliques 52

4 Vérification d'une situation critique locale 86

5 Recherche des répliques candidates 88

6 Recherche des répliques de référence candidates 89

7 Sélection de la réplique de référence 90

Introduction générale

Contexte

Réplication des données

Durant les dernières années, les volumes des données ne cessent de s'accroitre. Ceci est due à plusieurs facteurs tel que le streaming de données (les utilisateurs de YouTube téléchargent plus de 400 heures de vidéo par minute et nécessitent 1 péta-octet de stockage chaque jour [1]) ou le partage sur les réseaux sociaux (Chaque semaine, Facebook nécessite 60 To de stockage supplémentaire pour les nouvelles photos [2]). Ces données ne peuvent être stockées de manière centralisée, mais d'une manière distribuée, ce qui rend l'utilisation des systèmes à large échelle incontournable pour des applications qui utilisent de grands volumes de données. Parmi ces systèmes à large échelle, on peut noter les systèmes parallèles et distribués, les systèmes peer-to-peer (P2P), les systèmes de grille de données et, plus récemment, les systèmes Clouds. D'ici 2020, environ 40% des données de l'univers numérique seront stockées ou traitées par des fournisseurs de Cloud Computing [3].

Cloud Computing

Les systèmes Clouds sont des systèmes de calcul parallèle et distribué à grande échelle qui fournissent des ressources informatiques et de stockage évolutives via

Internet [4]. Les ressources peuvent être fournies en tant que services sans égard à l'endroit où elles sont fournies et à la manière dont elles sont fournies, similaire aux services d'eau, d'électricité, de gaz et de téléphonie [5]. Les types de services Cloud peuvent être classifiés en trois types de services : infrastructure (IaaS), plateforme (PaaS) et logiciel (SaaS). De plus, les clients Cloud peuvent louer les ressources de calcul et de stockage sur demande et ne payer que pour leurs consommations. Le Cloud Computing offre aux organisations qui gèrent d'immenses volumes de données et qui disposent de ressources internes limitées, la possibilité d'acquérir les ressources selon leurs besoins sur demande, tout en minimisant les coûts. Ils n'auront pas besoin d'investir sur l'installations des nouvelles infrastructures informatiques, ni d'investir sur des ressources humaines pour maintenir et gérer de telles infrastructures.

Problématique

La croissance constante des volumes de données, introduit plusieurs défis tel que, le stockage, l'accès, la gestion et le traitement incluant la recherche des données. En plus, les systèmes de stockage de données Cloud doivent généralement fournir une qualité de service (QoS) fiable pour satisfaire les engagements contractuels SLA (Service Level Agrement). Les spécifications techniques d'un SLA sont généralement décrites dans un SLO (Service Level Objectives). Les SLOs sont considérés comme un moyen pour mesurer la qualité des services comme la disponibilité ou le temps de réponse. Dans ce contexte, La réplication de données est une technique largement utilisée dans les environnements distribués qui stocke des données sur plusieurs sites, elle fournit la disponibilité et la performance. Si un site de stockage tombe en panne, le système peut continuer à fonctionner en utilisant des données répliquées, ce qui augmente la disponibilité et la tolérance aux pannes. En même temps, comme les données sont stockées sur plusieurs sites de stockage. La réplication des données permet de rapprocher les données des utilisateurs pour diminuer la latence, ou de créer de nombreuses copies pour des données extrêmement populaires afin de répartir la charge sur différents serveurs. Ce qui permet d'augmenter l'efficacité du système, de réduire la consommation de la bande passante et d'améliorer l'évolutivité du système. Donc, la réplication des données est très utile pour offrir de bonnes performances d'accès. La réplication est largement utilisée dans les systèmes parallèles [6], base de données [7], mobile [8], et à grande échelle y compris les grilles [9], Peer to Peer [10]

et Cloud Computing [11].

Lorsque la donnée est fréquemment mise à jour, la maintenance des répliques peut dégrader les performances. En plus, maintenir plusieurs copies d'une même donnée peut poser des problèmes de cohérence lors des mises à jour. En effet, lorsqu'une copie est modifiée, la modification doit être propagée à toutes les autres copies du système afin de garantir leur cohérence. Des mécanismes de gestion de cohérence, également appelés protocoles de cohérence, doivent être présents dans tous les systèmes de réplication. Plusieurs modèles et protocoles de cohérence offrant différents niveaux de garantie/performance ont été proposés.

D'après le théorème de CAP [12], Brewer affirme qu'il est impossible pour un système distribué de fournir à la fois cohérence, disponibilité et support des partitions. Il est donc nécessaire de faire des compromis sur l'une des trois propriétés en fonction des contraintes des applications. Assurer une cohérence forte nécessite d'énormes efforts de synchronisation sur différents sites et expose ainsi les utilisateurs à des latences de réseau élevées. Cela affecte les performances et la disponibilité des données. Une alternative particulière qui est devenue très populaire est la cohérence faible. Elle permet une certaine divergence entre les répliques. Lorsque la cohérence garantie est faible, une demande d'accès ne reflète pas forcément toutes les modifications antérieures, par contre, ces modifications seront transmises au bout d'un temps fini.

Nous avons comme premier objectif de proposer une stratégie efficace de réplication dynamique de données dans les Cloud Computing. Il existe un certain nombre de travaux dans la littérature qui traitent la réplication des données dans les systèmes de Cloud [13], [14]. Des stratégies ont été proposées pour améliorer les performances [11], [6], réduire la consommation de bande passante [15], augmenter le niveau de disponibilité des données [16], [17] et l'équilibrage des charges [18]. Cependant, ces objectifs semblent être difficilement satisfaits simultanément. Par exemple, la réplication des données garantit la disponibilité. Mais, cela se fait au détriment des communications entre les sites qui surchargent le réseau et donc un effet sur les performances. De plus, la plupart de ces stratégies négligent le coût de réplication et ne prennent pas en compte le profit du fournisseur.

Contributions

Première contribution

Nous proposons une stratégie de réplication de données dynamique qui répond simultanément aux exigences de disponibilité et de performance tout en prenant en compte le profit du fournisseur (DRAPP). Dans notre stratégie, nous traitons les trois problèmes suivants :

1. Quelles données doivent être répliquées et quand répliquer pour répondre à la fois aux exigences des utilisateurs et au bénéfice économique du fournisseur ? par exemple, des données répliquées trop tôt ne réduiront pas le temps d'attente ou n'accéléreront pas l'accès aux données.
2. Combien de nouvelles répliques appropriées doivent être créées pour garantir une disponibilité minimale et de réaliser un profit économique raisonnable pour le fournisseur ? À mesure que le nombre de répliques augmente, la disponibilité des données augmente. Mais en même temps, le coût de création et le maintien de ces nouvelles répliques vont augmenter aussi de manière significative et donc faire des dépenses inutiles.
3. Où les nouvelles répliques doivent être placées ? Des données proches des requêtes des utilisateurs, peuvent diminuer la latence et de réduire la consommation de la bande passante. Des données distribuées d'une façon optimisée peuvent améliorer l'équilibrage de charge des différents serveurs.

Notre idée générale est de créer de nouvelles répliques en fonction des contraintes de disponibilité et de performance (SLO). La création de nouvelles répliques est déclenchée uniquement si la contrainte de disponibilité de la donnée n'est pas garantie ou si le temps de réponse d'une requête d'un client est supérieur au seuil de temps de réponse convenu dans le SLA. Certaines informations collectées, par exemple la charge de travail d'un nœud et la popularité des données, sont également utilisées lors de la prise de décision. Dans notre stratégie nous calculons le nombre de répliques minimum à créer pour maintenir une disponibilité élevée des données pendant que les performances du système sont améliorées. Le nombre de répliques requises est ajusté dynamiquement. Les requêtes des locataires sont classées selon différentes

régions. Chaque région est composée de sous-régions, elle-même composée de centres de données (DC), tandis qu'un niveau hiérarchique de bande passante est considéré. Un nombre minimal de répliques est estimé pour chaque région afin de maintenir une disponibilité de données élevée. Nous vérifions certaines conditions telles que le budget et le coût de la réplication. Le coût de la réplication doit être inférieur au budget des utilisateurs. En conséquence, les répliques sont distribuées en fonction de ce coût. Nous estimons les revenus et les dépenses du fournisseur lors de l'exécution des requêtes des utilisateurs. Le fournisseur doit avoir un bénéfice réel pour déclencher le processus de la réplication de données.

Deuxième contribution

Dans cette contribution nous traitons le problème de gestion de la cohérence des données répliquées dans le Cloud. La gestion de la cohérence a un impact important sur les systèmes de stockage distribué. Elle doit être réaliser d'une façon efficace afin de respecter les contraintes de performance, de disponibilité et de coût dans le Cloud. Dans notre deuxième contribution, nous avons proposé un modèle basé sur une approche optimiste en utilisant les vecteurs de version entre les répliques pour gérer la cohérence des données. La stratégie proposée est appelée AC2R. Nous avons amélioré AC2R par l'intégration d'un mécanisme de convergence et de résolution de conflits entre les répliques qui convient au Cloud Computing. Ce mécanisme est basé sur les différentes formes de négociation entre les services de cohérence locale (LCS). La collaboration entre les différents services de cohérence est la clé de notre approche. Nous l'avons utilisé pour tirer profit des différences et des similitudes entre les répliques. Chaque service de cohérence local transmet son vecteur de référence aux autres services de différentes régions ; ces vecteurs de références sont utilisés pour construire un vecteur de référence global VREF. Les services de cohérence local utilisent le vecteur de référence global pour détecter les convergences, les divergences et les conflits entre les répliques. En présence d'un conflit, les différents services de cohérence collaborent entre eux en utilisant le processus de négociation pour faire converger les répliques du même fichier vers une réplique de référence globale.

Afin d'évaluer les deux stratégies proposées, de réplication et de cohérence de données, nous avons effectué une série d'expérimentations en utilisant le simulateur

CloudSim. Ce dernier est un framework de simulation généralisé et extensible qui permet la modélisation, la simulation et l'expérimentation des nouvelles infrastructures du Cloud Computing et les services d'application [19]. Dans un premier temps, nous avons étendu le simulateur Cloudsim afin de supporter et gérer notre stratégie de réplication des données. Les résultats des expérimentations montrent que la stratégie de réplication proposée peut améliorer de manière significative à la fois la disponibilité et la performance tout en assurant la rentabilité des fournisseurs du Cloud. Nous avons étendu également le simulateur Cloudsim afin de prendre en charge la cohérence des répliques. La stratégie de cohérence de donnée implémentée a été comparée à d'autres stratégies. Les résultats obtenus montrent l'efficacité de notre approche en terme de temps d'exécution et aussi la capacité d'éliminer les situations de divergence et de conflits.

Organisation du manuscrit

Les travaux de recherches que nous avons mené sont synthétisés dans ce document qui est organisé en deux grandes parties, outre une introduction générale et une conclusion.

Partie I : Cette première partie gère le problème de réplication de données dans les systèmes à grande échelle. Elle comprend trois chapitres :

- Le premier chapitre présente un état de l'art sur la réplication de données. Il conduit à une étude théorique des stratégies de réplication dans les grilles et les Clouds.
- Le deuxième chapitre décrit la stratégie proposée de réplication de données dynamique qui répond simultanément aux exigences de disponibilité et de performance tout en prenant en compte le profit du fournisseur.
- Le troisième chapitre est consacré à la validation de notre stratégie de réplication de données ; Il détaille et interprète les résultats des différentes expérimentations réalisées.

Partie II : dans la deuxième partie nous traitons le problème de la cohérence de données dans les Clouds, cette partie comprend trois chapitres :

- Le chapitre 1 présente un état de l'art sur les stratégies de cohérence de données.

Il cite quelques stratégies de cohérence de données dans l'environnement de la grille et il fait une classification des stratégies les plus référencées dans les Clouds.

- Le chapitre 2 décrit la stratégie proposée pour contribuer à la résolution du problème de gestion de cohérence des données dans le Cloud Computing et il met en évidence les améliorons effectuées.
- Dans le chapitre 3, l'évaluation de notre stratégie proposée est décrite à l'aide des résultats d'expérimentation ainsi leurs interprétations.

Finalement, nous conclurons cette thèse en dressant un récapitulatif des différentes contributions que nous avons apporté aux problèmes de réplication et de cohérence des données dans les Clouds. Nous terminerons par relater quelques pistes de recherche qui nous semblent pertinentes par rapport à ces problèmes.

Première partie

Gestion de la réplication de données

CHAPITRE 1

Etat de l'art sur les stratégies de réplication de données

1.1 Introduction

La réplication est l'un des phénomènes les plus étudiés dans les environnements distribués, c'est un domaine d'intérêt depuis de nombreuses années dans les réseaux pair-à-pair [20], [21], [22], les réseaux ad-hoc [23], [24] et les réseaux maillés [8]. La réplication est une stratégie qui crée plusieurs copies de certaines données et les stocke sur plusieurs sites où les coûts de création, de maintenance et de mise à jour des répliques sont des problèmes importants et difficiles. Plus récemment, l'émergence de systèmes distribués à grande échelle tels que les grilles [9], [25], [26], [27] et les Cloud [28], [29], [30], [31] rend la réplication des données un domaine de recherche prioritaire une fois de plus, et donc de nombreux chercheurs se sont concentrés sur la réplication des données pour atteindre des niveaux élevés de qualité de service. Ce chapitre conduit à une étude théorique des stratégies de réplication dans les systèmes à grande échelles. Nous décrivons différentes stratégies et nous discutons leurs avantages et inconvénients.

1.2 Stratégies de réplication dans les grilles

Différents critères de classification sont souvent étroitement liés dans les enquêtes récentes sur les stratégies de réplication des données. Certains travaux s'intéressent

à la classification statique et dynamique [32], [33], tandis que d'autres travaux se focalisent sur la gestion de la réplication centralisée et décentralisée [34], [35], [36]. Dans [37], les auteurs traitent la classification selon des fonctions objectives. Un autre travail [38] propose une classification des stratégies de réplication dynamique par rapport à l'architecture de la grille.

1.2.1 Stratégie de réplication statique vs dynamique

Le premier classement est basé sur le caractère statique ou dynamique de la stratégie de réplication. Les stratégies de réplication peuvent être classées en deux groupes différents, à savoir la réplication statique et dynamique. Dans la réplication statique, toutes les décisions concernant la stratégie de réplication sont prises avant que le système ne soit opérationnel et ces décisions restent inchangées pendant le fonctionnement [32], [33], [39], [40]. Contrairement à la réplication statique, la réplication dynamique crée et supprime automatiquement les copies selon l'évolution des demandes des utilisateurs [41], [42], [43], [44]. Dans un environnement de grille non modifiable, où les nœuds ne rejoignent pas ou ne quittent pas la grille et le mode d'accès aux fichiers reste le même durant l'exécution, la réplication statique peut être un bon choix. La réplication statique n'introduit pas un surcoût dû à la gestion de la réplication. D'un autre côté, lorsqu'un scénario de réplication doit être reconfiguré périodiquement en fonction des propriétés de la grille dynamique, il entraîne une charge administrative importante et affecte l'évolutivité et l'utilisation optimale des ressources du système. Dans un environnement dynamique où les nœuds sont libres de rejoindre ou de quitter le système, et où les modes d'accès aux fichiers changent avec le temps, la réplication dynamique excelle la réplication statique en raison de sa nature d'adaptabilité.

1.2.2 Stratégies de réplication centralisé vs décentralisé

Un autre type possible de classification se base sur la prise de décision, Cette classification concerne principalement les stratégies dynamiques qui peuvent être mises en œuvre de manière centralisée ou décentralisée [34], [35], [36], [45], [46]. Dans les stratégies de réplication centralisée [47], [48], [49], [50], [51], un serveur central de répliques est nécessaire pour gérer le processus de réplication. Le serveur central collecte toutes les informations pour déterminer les fichiers à répliquer et les sites

où les placer. Ce qui conduit à une latence d'accès et une charge étendues sur ce serveur. Dans une stratégie de réplication décentralisée, une certaine synchronisation est nécessaire afin d'obtenir de meilleurs résultats [45], [52], [53]. La coordination d'un événement de réplication est généralement effectuée avec la collaboration d'un certain nombre de nœuds. À mesure que le système évolue, le surcoût de la communication entre les nœuds ne doit pas surpasser les avantages de la réplication.

1.2.3 Stratégies de réplication basées sur des fonctions objectives

R. Mokadem et A. Hameurlain proposent une autre classification selon la fonction objective de la stratégie de réplication [37]. Une fonction objective est une méthode générale d'évaluation de la performance du système. Elle sert de critère pour optimiser une stratégie de réplication. Les objectifs possibles d'une stratégie de réplication sont :

- Exploiter les différentes formes de localités de données en considérant la popularité des données [52] ;
- Profiter de la localité au niveau du réseau [45], [54] ;
- Maximiser les avantages économiques [55] ;
- Exploiter un modèle de coût afin de décider de répliquer ou non, tout en minimisant le coût de la réplication [56].

A – Stratégies de réplication basées sur la localité

La popularité des données a été initialement proposée par Gwertzman et Seltzer [57]. Elle constitue un paramètre important que la plupart des stratégies de réplication considèrent pour répliquer la donnée la plus demandée. Elle peut être exprimée par le nombre de demandes d'accès à la donnée, qui est calculé par le taux d'accès à cette dernière. Afin de gérer la variation du taux d'accès à la donnée, certains travaux [58] appliquent une collecte périodique d'accès aux données pour déterminer sa popularité, tandis que d'autres travaux [48] proposent l'utilisation des historiques d'accès aux données pour calculer rapidement leur popularité.

Les travaux de Ranganathan et Foster [52] sont les premiers à exploiter la popularité des données lors de la réplication des données dans les grilles. Ils ont proposé six

stratégies de réplication dynamique pour une grille de données à plusieurs niveaux. Ces stratégies sont les suivantes :

- Aucune réplication ou mise en cache n'est implémentée que pour comparer les autres stratégies à un scénario de non-réplication.
- Dans la stratégie du meilleur client, les enregistrements de l'historique des accès sont conservés pour chaque fichier de la grille. Lorsqu'un certain seuil est atteint, le fichier est répliqué uniquement sur le client qui génère la plupart des demandes.
- La réplication en cascade introduit une stratégie de réplication à plusieurs niveaux, dans laquelle, lorsqu'un seuil est dépassé pour un fichier au niveau du nœud racine, une réplique est placée progressivement dans le prochain niveau et sur le chemin vers le meilleur client.
- Dans la mise en cache simple, le client demande un fichier et le stocke localement.
- Mise en cache plus réplication en cascade combine la réplication en cascade et la mise en cache simple.
- Dans la stratégie Fast Spread, une réplique du fichier demandé est stockée sur chaque niveau tout au long de son chemin d'accès au client. La popularité et l'âge du fichier sont utilisés comme des paramètres pour sélectionner les répliques qui doivent être remplacées.

Les stratégies sont comparées selon trois modes d'accès différents, les résultats montrent que la stratégie du meilleur client est la moins performante. Fast Spread fonctionne mieux avec les modes d'accès aux données aléatoires et la réplication en cascade fonctionne mieux lorsque la localité existe dans les modes d'accès aux données.

Ils ont introduit trois types de localités différentes [52], à savoir :

- La localité temporelle dans laquelle une donnée qui a été accédée récemment est beaucoup plus susceptible d'être demandée prochainement.
- La localité géographique dans laquelle un fichier qui a été récemment consulté par un client, doit probablement être demandée par les clients à proximité (le modèle hiérarchique de la grille reflète généralement la localité géographique).

– La localité spatiale dans laquelle les fichiers associés au fichier récemment consulté seront probablement demandés dans un futur proche.

Stratégies de réplication basées sur la localité temporelle et géographique

Tang et al. [44] ont proposé deux algorithmes : Simple Bottom-Up (SBU) et Aggregate Bottom-Up (ABU) pour une architecture de grille de données à plusieurs niveaux. L'idée générale est d'exploiter la localité géographique et temporelle en plaçant des répliques aussi proches que possible du client en se basant sur leur popularité. L'algorithme SBU réplique le fichier qui dépasse un seuil prédéfini. Cependant, SBU ne prend pas en considération la relation entre les enregistrements d'historiques d'accès. Afin de résoudre ce problème, ABU est conçu pour agréger les enregistrements d'historiques au niveau supérieur jusqu'à ce qu'il atteigne la racine. Les résultats de la simulation montrent que l'utilisation d'ABU réduit le temps de réponse moyen et le coût moyen de la bande passante par rapport aux solutions SBU et Fast Spread, en particulier lorsque la taille de stockage disponible des serveurs est très faible.

Wu et al. [59] s'intéresse à assurer un équilibrage de charge entre les répliques en proposant un algorithme de placement qui trouve les serveurs optimaux pour les répliques. Les auteurs considèrent le problème d'une localité géographique. Par conséquent, un utilisateur peut spécifier la distance minimale qui le sépare du serveur de données le plus proche afin de créer la réplique.

Bsoul et al. [51] ont proposé une amélioration de la stratégie de réplication Fast Spread EFS (Enhanced Fast Spread) pour une architecture générale d'une grille. Lors de la prise de décision de réplication, les paramètres comme le nombre et la fréquence des demandes, la dernière fois où la réplique a été demandée et la taille de la réplique sont pris en considération. Les résultats de la simulation montrent que l'EFS fonctionne mieux que la stratégie originale (Fast Spread).

Nukarapu et al. [60] ont proposé une stratégie de réplication des données qui a une garantie de performance prouvée théoriquement. Le point clé de cette stratégie est que lorsque plusieurs répliques sont disponibles, chaque nœud conserve la réplique la plus proche géographiquement. Les résultats de la simulation montrent que cette stratégie surpasse de manière significative la stratégie de réplication populaire existante sous divers paramètres de réseau.

Stratégies de réplication basées sur la localité spatiale

La plupart des travaux mentionnés ci-dessus sont concentrés sur la localité temporelle et géographique et ils ont négligé la localité spatiale. Ceci est expliqué par le fait que la réplication est généralement effectuée après l'arrivée des requêtes qui entraînent un retard important. Afin de réduire ce délai, la réplication doit être effectuée à l'avance.

Dans ce contexte, Chang et al. [49] ont abordé certains problèmes de stratégies de réplication basées sur la localité temporelle et se sont également concentrés sur les problèmes de déplacement de données en prédisant les besoins futurs des fichiers, c'est-à-dire la localité spatiale. Grâce à une méthode prédictive, le temps d'exécution du travail est réduit en effectuant une mise en cache des fichiers susceptibles d'être demandés ultérieurement.

Lei et Vrbsky [48] ont également abordé ce problème en proposant l'algorithme LAWL (Last Access Largest Weight) pour une grille de données à plusieurs niveaux. En plus de la localisation temporelle lors de la détermination de la popularité des fichiers, différents poids sont attribués aux fichiers en fonction de leur âge, ce qui augmente l'importance des nouveaux fichiers. En conséquence, il donne une mesure plus précise pour déterminer un fichier populaire pour la réplication et le nombre de répliques. Toutefois, le placement des répliques est effectué uniquement au niveau du cluster et non au niveau du nœud. De plus, certains travaux de recherche [61] l'ont classé comme une méthode centralisée en raison de la présence d'un Cluster-Head qui collecte les informations d'accès aux fichiers de tous les autres Clusters-Head.

Khanli et al. [61] ont étendu la stratégie de propagation rapide, proposée par [52], en proposant la méthode PHFS (Predictive Hierarchal Fast Spread) conçue pour réduire la latence d'accès aux données dans les systèmes de grille de données hiérarchiques. Il utilise des techniques prédictives pour prédire l'utilisation future des fichiers. Ensuite, il les pré-réplique sur un chemin de la source au client, c'est-à-dire l'utilisateur qui travaille dans le même contexte peut demander des fichiers avec une probabilité élevée à l'avenir.

Stratégies de réplication basées sur la localité au niveau du réseau

La plupart des stratégies de réplication existantes essaient de maximiser la localité de données afin de réduire le temps d'accès aux données. Cependant, la capacité de stockage de chaque nœud peut être limitée. Des grands volumes de données sont créés par les systèmes à grande échelle mais seulement une petite partie de ces données peut être prise en charge par les nœuds de la grille. En conséquence, l'effet de la localité de données est réduit. Certains travaux de recherche tirent parti d'une autre forme de localité, appelée « localité au niveau du réseau ». Ce type de localité indique que le fichier demandé est situé sur le nœud qui a la bande passante la plus large. En conséquence, la congestion du réseau est l'une des fonctions objectives à optimiser.

Dans ce contexte, Park et al. [54] ont proposé une stratégie de réplication dynamique basée sur une bande passante hiérarchique appelée BHR (Bandwidth Hierarchy based Replication). La stratégie proposée bénéfice de la localité au niveau du réseau afin de réduire le temps d'accès aux données en évitant la congestion du réseau dans les grilles de données. Ils ont classé les nœuds en plusieurs régions. Ensuite, la bande passante entre les nœuds dans une même région est plus large que la bande passante entre les nœuds des régions différentes. Puisque la bande passante dans la région est plus grande, BHR essaye de maximiser le nombre de données requises dans la même région afin de récupérer la réplique plus rapidement. Dans ce contexte, une popularité régionale des fichiers est considérée. Cependant, la stratégie BHR n'a de bonnes performances que lorsque la capacité de stockage est faible.

D'autres travaux de recherche [62] ont utilisé l'algorithme BHR pour traiter les problèmes d'ordonnancement et de réplication. En adoptant cette stratégie, les auteurs affirment que la décision de réplication est faite pour l'optimisation à long terme. L'algorithme proposé produit de bons résultats surtout lorsque la hiérarchie de bande passante est simple.

Plus tard, Sashi et Thanamani [45] ont proposé un algorithme BHR modifié pour surmonter les limites de l'algorithme BHR standard. L'algorithme BHR modifié augmente la disponibilité des données en répliquant un fichier dans le nœud où a été fréquemment consulté. Cela permet d'éviter la réplication inutile des fichiers et donc d'utiliser le réseau plus efficacement. En conséquence, moins de temps sera consommé

pour récupérer le fichier demandé si ce dernier se trouve dans la même région locale. Cependant, la recherche du meilleur nœud entre tous les nœuds constitue la principale faiblesse de l'algorithme BHR modifié. En outre, les données peuvent ne pas toujours être présentes dans les emplacements proches avec une bande passante élevée dans l'environnement de la grille de données.

B – Stratégies de réplication basées sur des comportements économiques

La stratégie de gestion des répliques économiques a été introduite par Sidell et al [63] dans le système Mariposa. Les auteurs utilisent des fonctions d'évaluation pour décider de créer ou non des répliques locales. Ils ont exploité le concept des "enchères" pour sélectionner la meilleure réplique pour une tâche en utilisant les modes d'accès aux fichiers. Un courtier de stockage participe à ces enchères en proposant un prix auquel il vendra l'accès à une réplique si elle est disponible. Sinon, il lance une enchère pour répliquer le fichier demandé sur ses sites de stockage s'il détermine que cela est économiquement réalisable.

La stratégie de réplication de Carman et al. [64] repose également sur le même principe. Chaque nœud tente d'acheter un élément de données pour créer la réplique sur son propre nœud. La valeur d'un fichier est calculée comme la somme du futur paiement qui sera reçu par un nœud. Cela permet de générer des revenus à l'avenir en les vendant à d'autres nœuds. Les auteurs se sont concentrés sur l'optimisation de la réplication afin de réduire le temps d'exécution à long terme. Ils montrent une amélioration significative par rapport aux stratégies de réplication traditionnelles.

Lin et al. [65] ont également proposé une stratégie de réplication basée sur les comportements économiques. Un courtier de réplication est utilisé pour réduire les surcoûts des mécanismes de réplication afin de prendre en compte les politiques concernant le transfert de données.

Enfin, Andronikou et al. [66] ont présenté une stratégie de réplication de données prenant en compte la qualité de service pour une architecture centralisée. Il détermine le nombre de répliques nécessaire tout en tenant compte les contraintes d'infrastructure, telles que l'équilibrage de charge sur tous les nœuds, la bande passante et l'importance de données. Un ensemble de données important est celui pour lequel il est impératif de maintenir les garanties aux exigences de QoS, soit parce

qu'il produit plus de profit soit parce qu'il génère le plus grand déficit en cas de violation des garanties de QoS.

C – Stratégies de réplication basées sur des modèles de coûts

Les stratégies de réplication basées sur des modèles de coûts traitent l'estimation des gains d'accès aux données ainsi que l'estimation des coûts de création et de maintenance des répliques. Le calcul de ces estimations repose sur la latence du réseau, la bande passante et la taille de la réplique.

Ranganathan et al. [48] ont proposé une stratégie de réplication pour les systèmes de grille de données basés sur une topologie P2P. Chaque Peer réagit indépendamment des autres pour prendre une décision de réplication afin d'améliorer la disponibilité des données. Le Peer qui maximise la différence entre le coût total et le bénéfice de la réplication est considéré comme le meilleur client. L'avantage de cette stratégie est qu'il n'y a pas de point de défaillance unique. Cependant la limite de cette stratégie réside dans le fait que les auteurs ont supposé une quantité illimitée de stockage ce qui n'est pas réaliste. De plus, cette stratégie ne tient pas compte de l'état de la grille et nécessite un nombre minimum de répliques.

Chervenak et al. [32] ont proposé un algorithme de réplication basé sur un modèle de coût pour les systèmes de grille de données hiérarchiques de type arbre. Il utilise un modèle de coût pour prédire si les répliques valent la peine d'être créées ou pas. Les résultats de la simulation ont montré qu'il est préférable que les feuilles (représentent des clients) exécutent des tâches et que les nœuds supérieurs contiennent toutes les ressources de stockage. Bien que cette stratégie soit très prometteuse, le problème réside dans le fait que les résultats ne sont comparés qu'au cas où aucune réplication n'a été effectuée.

L'utilisation d'un modèle d'estimation des coûts dans les stratégies de réplication a également été bien exploitée par Lamehamedi et al. [56] pour une topologie de grille de données hybride. Afin de décider si la réplication doit être effectuée ou non, l'amélioration d'accès aux données obtenue par la réplication (bénéfice) est comparée au coût de création d'une réplique et à sa maintenance pendant l'exécution. Une fonction de coût est utilisée pour classer les fichiers dans le site de stockage local. Puis, un gestionnaire de réplication ne crée une nouvelle réplique que si elle améliore

le coût du transfert de données. Les paramètres qui sont considérés avant de créer et de placer une réplique sont : les modes d'accès, la capacité de stockage disponible et les estimations des coûts citées ci-dessus. Les expériences montrent que les gains de performance augmentent avec la taille des données.

Plus tard, Zhang et al. [67] construisent un modèle probabiliste pour la grille de données hiérarchique afin de prédire ses performances optimales. Ils montrent que l'algorithme de réplication optimale (ORA) proposé est meilleur par rapport aux trois stratégies de réplication (ABU, SBU et Fast Spread).

1.2.4 Stratégies de réplication dynamique selon l'architecture

La topologie de la grille a un impact important sur la stratégie de réplication. En effet, une stratégie de réplication proposée dans le cadre d'une grille hiérarchique ne produira pas les mêmes résultats pour une grille complètement distribuée.

Tos et al. [38] proposent une nouvelle classification des stratégies de réplication dynamique par rapport à l'architecture de la grille. Les stratégies sont classées selon trois grandes architectures : des stratégies dédiées aux grilles hiérarchiques, stratégies dédiées aux grilles P2P et enfin les stratégies de réplication de données pour les architectures hybrides.

A – Stratégies de réplication dynamique pour les architectures hiérarchiques

Dans la littérature, la plupart des chercheurs ont travaillé sur les structures hiérarchisées [32], [33], [51], [55], [66], [68] et ont mentionné l'extension de leurs recherches aux topologies sous forme de graphe. Les architectures hiérarchiques sont généralement des réseaux structurés sous la forme d'un arbre ou d'une étoile.

Stratégies de réplication dynamique pour une architecture à plusieurs niveaux

Shorfuzzaman et al. [69] proposent deux stratégies de réplication dynamique pour la grille de données à plusieurs niveaux. Dans la première stratégie appelée PBRP (Popularity Based Replica Placement), le placement de la réplique est basé sur la

popularité. La deuxième stratégie nommée PBRP adaptatif (APBRP) est une amélioration de la première stratégie. PBRP vise à équilibrer le compromis entre la latence d'accès et l'utilisation du stockage en répliquant les fichiers en fonction de la popularité des fichiers. La stratégie de réplication est lancée périodiquement de sorte que les enregistrements d'accès soient agrégés de bas en haut et que le placement des répliques se fasse de haut en bas. La version améliorée (APBRP) introduit un seuil de taux d'accès adaptatif. Dans les simulations, les deux stratégies donnent des résultats meilleurs que les stratégies Best Client, Cascading, Fast Spread et ABU en termes de temps d'exécution des tâches, d'utilisation moyenne de la bande passante et d'utilisation du stockage. Les résultats de la stratégie APBRP sont meilleurs par rapport à PBRP.

Abdurrab et Xin [70] présentent une stratégie de réplication, appelée FIRE (FIle REunion), qui tire parti de la corrélation entre les demandes de fichiers. FIRE suppose qu'il existe une forte corrélation entre un groupe de tâches et un ensemble de fichiers. Sur la base de cette hypothèse, FIRE vise à réunir l'ensemble de fichiers sur les sites en utilisant la réplication. La réplication est effectuée lorsqu'un fichier n'est pas disponible localement et qu'il y a suffisamment d'espace de stockage pour le stocker. Si l'espace de stockage est insuffisant, un fichier avec un degré de corrélation de groupe inférieur est supprimé avant la réplication du nouveau fichier. Dans un scénario de simulation, FIRE a une meilleure performance que les stratégies de réplication LFU (Least Frequently Used) [71] et LRU (Least Recently Used).

Stratégies de réplication dynamique basée sur la hiérarchie de la bande passante

Mansouri et Dastghaibyfard [46] ont étendu la stratégie 3LHA (3-Level Hierarchical Algorithm) [72] et ont proposé l'algorithme de réplique hiérarchique dynamique nommé DHR (Dynamic Hierarchical Replication). Ils soulignent que 3LHA place des répliques dans tous les sites demandeurs. D'un autre côté, DHR crée une liste ordonnée de sites par région en fonction du nombre d'accès à un fichier. Le premier site dans la liste ordonnée est choisi pour héberger la nouvelle réplique. En plaçant des répliques sur les meilleurs sites, DHR vise à réduire les coûts de stockage et le temps d'exécution des tâches. Ils comparent l'efficacité de DHR avec les stratégies

suivantes : aucune réplication, LFU, LRU, BHR et 3LHA. Les résultats montrent que DHR donne de meilleurs temps d'exécution de travail comparé aux autres stratégies citées, particulièrement quand les sites de la grille ont un espace de stockage plus petit.

Dans un autre article, Mansouri et Dastghaibyfard [73] ont proposé une amélioration de leur stratégie de réplication hiérarchique dynamique nommée EDHR (Enhanced DHR), ils ont ajouté un modèle de calcul de coût économique à la DHR. En estimant la future valeur économique des fichiers, ils ont fait une évaluation des répliques qui ne seront pas bénéfiques et donc seront supprimées, et quels seront les fichiers bénéfiques qui seront répliqués. Les simulations indiquent que EDHR produit des temps d'exécution encore meilleurs que la DHR.

Mansouri et al. [74] proposent également un algorithme de réplication hiérarchique dynamique modifiée appelé MDHRA (Modified Dynamic Hierarchical Replication Algorithm), qui est une autre extension de la stratégie DHR. Dans MDHRA, le mécanisme de décision de placement de la réplique est modifié pour prendre en compte le temps de la dernière demande, le nombre d'accès et la taille de la réplique. Ils notent que la nouvelle approche améliore la disponibilité de répliques. Les simulations montrent que, par rapport à la DHR et aux autres stratégies étudiées, MDHRA fonctionne mieux en termes de temps de réponse moyen et elle permet une utilisation efficace du réseau. Cependant, les comparaisons de performance n'incluent pas EDHR.

Stratégies de réplication dynamique pour d'autres architectures hiérarchiques

Perez et al. [75] présentent BRS (Branch Replication Scheme) qui a trois caractéristiques principales : (i) la création de sous-répliques optimise l'utilisation du stockage, (ii) les performances d'accès aux données sont augmentées via les E/S parallèles et (iii) la cohérence des mises à jour pour autoriser la modification des répliques. Dans BRS, les fichiers sont divisés en plusieurs sous-répliques disjoints placés sur des nœuds différents. Avec cette approche, BRS vise à créer des niveaux élevés de tolérance aux pannes sans augmenter l'utilisation du stockage. Dans les simulations, BRS est comparée à la stratégie de réplication hiérarchique et montre

des performances d'accès aux données améliorées sur les opérations de lecture et d'écriture.

Zhao et al. [76] proposent une stratégie de réplication appelée DORS (Dynamic Optimal Replication Strategy). Un fichier est répliqué lorsque le nombre de répliques de ce fichier est inférieur à un seuil. Ce seuil est le rapport entre la capacité de stockage totale de la grille et la taille totale des fichiers sur la grille. Pour la politique de placement des répliques, ils ont conçu un modèle qui calcule la valeur des répliques. Lorsqu'une insuffisance d'espace de stockage est détectée, les répliques ayant la valeur la plus faible sont remplacées. La valeur de réplique dépend de la fréquence d'accès et du coût d'accès aux répliques. Les résultats de comparaison de DORS avec LFU et LRU montrent que DORS fonctionne mieux en termes de temps d'exécution moyen des tâches et garantie une utilisation efficace du réseau.

Meroufel et Belalem [77] proposent une stratégie de réplication, appelée Placement Dynamic (PD). Le but de PD est de minimiser le nombre de répliques pour assurer un certain degré de disponibilité sans dégradation des performances. Dans la stratégie proposée, le placement des répliques et les pannes dans le système sont pris en compte. Si une suspicion de panne est observée, les données seront déplacées vers d'autres nœuds du système pour maintenir le niveau de disponibilité. Les auteurs ont comparé la PD avec une approche de réplication aléatoire via des simulations effectuées avec FTSim. Les résultats montrent que PD assure de meilleurs temps de récupération et garantie la disponibilité désirée par rapport à la réplication aléatoire.

B – Stratégies de réplication dynamique pour une architecture peer-to-peer

Dans les architectures P2P, les nœuds agissent de manière autonome, sans intervention d'une autorité centrale. Les nœuds possèdent suffisamment de fonctionnalités pour être serveurs et clients en même temps. Cette structure décentralisée permet une volatilité plus élevée que les autres architectures, car les nœuds peuvent se connecter à n'importe quelle partie de la grille et partir sans préavis. Les stratégies de réplication pour l'architecture P2P sont développées en gardant à l'esprit cette nature dynamique des grilles P2P [78].

Bell et al. [55] ont présenté une stratégie de réplication de données basée sur l'éco-

nomie pour les grilles de données P2P. Dans la stratégie proposée, la grille de données est traitée comme un marché, où les fichiers représentent les biens qui sont échangés par les agents d'optimisation du système. Les éléments informatiques achètent des fichiers et visent à minimiser les coûts d'achat. De même, les éléments de stockage essaient de maximiser leurs profits et de faire des investissements basés sur les prédictions d'accès aux fichiers pour augmenter les revenus.

Abdullah et al. [79] proposent deux stratégies de réplication dynamique, la stratégie de placement selon le chemin et le nœud demandeur et la stratégie de placement de nœud de distance N-hop. La stratégie de placement selon le chemin et le nœud demandeur réplique les fichiers dans tous les nœuds qui se trouvent dans le chemin entre le nœud qui contient la réplique et le nœud demandeur, y compris le nœud demandeur. Dans la stratégie de placement des nœuds de distance N-hop, les répliques sont placées sur tous les voisins du nœud fournisseur avec une distance de N. Les résultats de simulations montrent que les stratégies proposées augmentent la disponibilité et réduisent le temps de réponse mais avec une utilisation énorme de la bande passante.

Challal et Bouabana-Tebibel [80] ont présenté une stratégie de réplication des données a priori pour les systèmes de grille de données P2P. Leur stratégie complète la réplication dynamique en trouvant des nœuds optimaux pour placer les répliques initiales avant le début des travaux. En maximisant la distance entre les répliques d'une même donnée et en minimisant les distances entre les différentes répliques des données. La stratégie proposée permet d'augmenter la disponibilité et de garantir que chaque nœud possède des répliques de fichiers différents dans son voisinage. Dans leurs simulations, une stratégie de placement d'une réplique a priori est comparée à un placement initial de répliques aléatoire et à avec aucun placement initial de répliques. La stratégie proposée améliore le temps d'exécution des tâches et réduit le temps de transfert des fichiers sans augmenter les coûts de stockage, ni l'utilisation de la bande passante.

Chettaoui et Charrada [81] proposent DPRSKP (Decentralized Periodic Replication Strategy based on Knapsack Problem) qui est une stratégie de réplication périodique décentralisée basée sur le problème du sac à dos. Deux caractéristiques

principales de cette stratégie sont distinguées : l'hypothèse de stockage limité pour les sites de grille et la dynamicité de la grille de données, c'est-à-dire le nombre de sites de grille qui existent à un moment donné. DPRSKP sélectionne les éléments à répliquer en créant une liste hiérarchisée en fonction de la popularité et de la disponibilité de chaque fichier. Les répliques de fichiers populaires sont ensuite placées sur des nœuds stables et ayant une bonne bande passante vers les nœuds demandeurs. Cet objectif est accompli en le formulant et en le résolvant comme le problème du sac à dos.

C – Stratégies de réplication dynamique pour une architecture hybride

Dans cette sous-section, nous avons étudié les stratégies de réplication de données dynamiques pour les architectures hybrides. Les architectures de grille de données hybrides combinent généralement au moins deux autres architectures avec des propriétés différentes.

Lamehamedi et al. [82] présentent une stratégie de réplication hybride combinant l'architecture hiérarchique avec les fonctionnalités P2P. Ils ont mis en place un modèle de coût, les décisions de réplication sont basées sur la façon dont les gains de la réplication mesurent les coûts. Un composant d'exécution surveille en permanence la grille pour collecter des paramètres importants, à savoir la taille de la réplique et l'état de la grille. Ces informations sont utilisées dans le calcul des coûts de réplication. Ils ont évalué trois scénarios de simulation différents sur une seule architecture. Les résultats indiquent que le temps de réponse moyen est amélioré à mesure que les répliques sont placées plus près des clients.

Rasool et al. [49] ont proposé une stratégie réplication bidirectionnelle appelé TWR (Two-Way Replication) qui combine une architecture multi-tiers avec des fonctionnalités similaires à P2P. Dans cette architecture, en plus d'être connecté au nœud parent, chaque nœud (sauf au niveau feuille) est également connecté à ses voisins du même niveau. La décision de réplication est gérée par une autorité centrale, appelée Grid Replication Scheduler (GRS). Les GRS ciblent les fichiers dont la fréquence d'accès est supérieure à la moyenne, ces fichiers sont répliqués sur le parent du client qui a fait le plus de demandes. Les fichiers avec des fréquences d'accès faible sont répliqués au niveau des grands-parents. Une étude de simulation montre que, TWR

s'est comporté de manière similaire à Fast Spread en termes de temps de réponse, tout en consommant moins de ressources.

1.3 Stratégies de réplication dans le Cloud Computing

1.3.1 Statique vs dynamique

Milani et al ont présenté une classification des techniques de réplication de données dans le Cloud [13], ils ont classé les stratégies de réplication en deux groupes principaux, y compris les mécanismes de réplication statiques et dynamiques. Les stratégies de réplication statique suivent des politiques déterministes [83]. Elles sont simples à mettre en œuvre mais elles ne sont pas souvent utilisées car elles ne s'adaptent pas en fonction de l'environnement [84]. Dans les modèles statiques, le nombre de répliques créées est constant et fixe depuis le début. Ce nombre est déterminé par l'utilisateur ou par l'environnement du Cloud.

Dans une stratégie de réplication statique, le nœud hôte et le nombre de répliques sont prédéterminés et bien définis [85], [86], [87] ; alors que les stratégies dynamiques créent et suppriment automatiquement les répliques en fonction des changements dans le modèle d'accès de l'utilisateur, la capacité de stockage et la bande passante [18], [41], [88]. Il fait des choix intelligents sur la localisation des données en fonction des informations de la situation actuelle. Mais, il a quelques inconvénients tels que la difficulté de recueillir des informations d'exécution de tous les nœuds de données dans une infrastructure de Cloud complexe et difficile à maintenir la cohérence du fichier de données [83]. Les algorithmes de réplication statique et dynamique peuvent être classés en algorithmes distribués [18], [89] et centralisés [17], [53], [85].

1.3.2 Stratégies de réplication de données selon les fonctions objectives

La technique de réplication de données est largement adoptée dans le système de Cloud Computing afin de garantir une haute disponibilité des données [17], [90], [91], [92], d'augmenter la fiabilité [4], [5], [20], [88], [93], [94], [95] ou d'améliorer le niveau de performance du système [7], [16], [96], [97], [98], [99], [100], [101], [102].

Nous proposons une classification des stratégies de réplications de données dans le Cloud en se basant sur les objectifs cités ci-dessus.

A – Disponibilité des données

Dans cette partie, nous présentons quelques travaux de réplication de données qui ont pour objectif d'améliorer la disponibilité des données dans les Clouds Computing.

La plus part des services de stockage existants dans le Cloud sont conçus et construits sur l'hypothèse où toutes les ressources de stockage constituent un ensemble homogène de ressources distribuées. Cette hypothèse conduit ces systèmes à considérer une disponibilité unique pour tous les nœuds du Cloud, puis à optimiser leur redondance de données. Cependant, cette hypothèse simplifie la façon dont la disponibilité des données est mesurée, mais elle introduit une erreur qui entraîne une augmentation de la redondance des données et une perte d'efficacité. Dans le travail [91], les auteurs ont étudié comment la non-prise en compte des hétérogénéités des nœuds affecte négativement les performances des infrastructures Cloud hétérogènes. Pour ce faire, ils ont présenté un framework analytique qui a trois avantages principaux : *i*) un algorithme de mesure de la disponibilité des données dans des infrastructures de stockage hétérogènes ; *ii*) un algorithme d'optimisation pour trouver le meilleur moyen d'affecter des blocs redondants à l'ensemble des nœuds de stockage ; *iii*) un mécanisme pour déterminer la redondance minimale des données afin d'obtenir une qualité de service souhaitée.

L'un des principaux résultats de ce framework est la fonction d'attribution de données. Les meilleurs résultats sont obtenus lorsque les nœuds reçoivent une redondance proportionnelle à leurs disponibilités. Cette solution peut réduire la redondance des données jusqu'à 70% dans des scénarios très hétérogènes. Ces résultats montrent comment l'hétérogénéité est un aspect important à considérer dans les infrastructures de stockage distribué en général, et dans les Clouds hétérogènes en particulier.

Une architecture hiérarchique à plusieurs niveaux est utilisée par Gill and Singh [84], Ils ont proposé une stratégie appelée DCR2S (Dynamic Cost-aware Replication and Re-balancing Strategy). Cette stratégie est basée sur le coût de réplication et de rééquilibrage pour un système Cloud hétérogène. DCR2S détermine quel fichier doit être répliqué et quand le répliquer. Elle détermine également le nombre approprié de

répliques. DCR2S permet d'optimiser le coût de la réplication en utilisant le concept du problème de sac à dos. Une fois que le coût de la réplication dépasse le budget de l'utilisateur, les données placées dans un centre de données avec un coût élevé sont répliquées à nouveau dans des centres de données à moindre coût. La réplication est effectuée jusqu'à ce que la disponibilité soit supérieure ou égale à la disponibilité souhaitée.

De même, pour un environnement de Cloud hétérogène et hautement dynamique, Mohamed-K Hussein et Mohamed-H Mousa [90] ont proposé une stratégie de réplication de données qui sélectionne de façon adaptative les fichiers de données pour la réplication afin d'améliorer la disponibilité globale du système et de répondre à la qualité de service requise. En outre, la stratégie proposée détermine dynamiquement le nombre des nouvelles répliques de manière heuristique en améliorant la disponibilité de chaque fichier. Ils utilisent un algorithme légé de série temporelle pour prédire les futures demandes aux fichiers. La décision de réplication est principalement basée sur les prédictions fournies. Une fois que le facteur de réplication qui est basé sur la popularité des fichiers est inférieur au seuil spécifique, la réplication est déclenchée. La stratégie proposée identifie le meilleur placement des répliques en utilisant une recherche heuristique. L'heuristique proposée est peu coûteuse et peut gérer des ressources et des données à grande échelle dans un délai raisonnable.

La modélisation d'une stratégie de placement de données pour améliorer la disponibilité peut également être réalisée à l'aide de modèles de coûts économiques personnalisés. Certains travaux présentent un modèle d'enchères pour implémenter une stratégie de placement de répliques dans un environnement de stockage de Cloud [92]. Si le niveau de disponibilité souhaité ne peut pas être maintenu, une enchère sera lancée pour déterminer le placement d'une nouvelle réplique. Le prix de l'enchère dépend de plusieurs propriétés des nœuds, notamment la probabilité de défaillance, la bande passante du réseau et l'espace disponible.

B – Fiabilité des données

Dans les systèmes distribués à grande échelle, en raison de la grande quantité de dispositifs de stockage utilisés, les défaillances de ces dispositifs se produisent fréquemment [3]. Par conséquent, l'importance de la fiabilité des données est cruciale,

et ces systèmes ont besoin d'une meilleure conception et gestion pour faire face aux défaillances fréquentes. L'augmentation du niveau de redondance des données pourrait être un bon moyen pour augmenter la fiabilité des données [4], [5] . En effet, la réplication de données est actuellement l'approche la plus populaire dans les systèmes de stockage distribués qui vise à augmenter le niveau de redondance des données. Parmi les travaux qui s'intéressent à augmenter la fiabilité des données dans les Cloud Computing nous citons : Li et al. [88], qui ont proposé une stratégie de réplication de données pour les applications basées sur le Cloud. Cette stratégie est appelée CIR « Costeffective Incremental Replication ». L'objectif principal de CIR est de gérer de manière rentable le problème de fiabilité des données dans un centre de données. Dans CIR, une stratégie de réplication incrémentale est proposée pour calculer le temps de création de réplique, qui indique la durée de stockage à laquelle l'exigence de fiabilité peut être satisfaite. Elle permet de savoir quand une réplique supplémentaire est nécessaire pour garantir la fiabilité requise. Le compromis entre le coût et la performance est un problème qui n'a pas été résolu simultanément.

Dans un autre travail, Li et al. [93] ont présenté un mécanisme de gestion de fiabilité des données rentable basé sur la vérification proactive des répliques nommée PRCR (Proactive Replica Checking for Reliability) pour réduire la consommation d'espace de stockage, d'où le coût de stockage des applications qui consomment beaucoup de données dans le Cloud. L'objectif de PRCR est de réduire le nombre de répliques stockés dans le Cloud tout en répondant aux exigences de fiabilité des données. PRCR a les caractéristiques suivantes : i) PRCR est capable de fournir une assurance de fiabilité des données au niveau du fichier. ii) PRCR est capable de fournir une gestion de fiabilité des données d'une manière plus rentable. PRCR permet d'assurer une large gamme de fiabilité des données avec au plus deux répliques. Cependant, ce travail ne considère que les facteurs de coût et de fiabilité pour le stockage des données et il n'a pas discuté les implications sur la performance d'accès aux données en utilisant PRCR. Une amélioration de PRCR a été proposée par Li et al dans le travail [94]. Ils ont considérablement étendu le modèle de fiabilité des données proposé dans [93]. Dans le nouveau travail, les auteurs ont étudié un cas particulier du modèle de fiabilité des données en supposant que le taux de défaillance d'un disque est constant et ils ont montré que la gestion de la fiabilité des données

avec un taux de défaillance d'un disque variable est faisable.

Un algorithme de réplication de données pour les Clouds de stockage nommé RFH (Resilient, Fault-tolerant et High-efficient) a été proposé dans l'article [95], qui est un algorithme tolérant aux pannes et adapté à la réplication globale. RFH utilise un schéma de migration flexible avec un algorithme de réplication pour modifier dynamiquement le nombre et l'emplacement des répliques afin de répondre à différents besoins. Il adopte également une fonction de suppression pour récupérer les ressources inutilisées. Le concept d'anneau virtuel et de nœud virtuel est utilisé dans l'algorithme RFH. Chaque partition de données est représentée par un nœud virtuel. Chaque nœud virtuel décide lui-même s'il doit répliquer, migrer ou supprimer selon un agent de décision. L'agent de décision utilise l'évaluation de la charge de trafic de tous les nœuds afin de sélectionner un nœud physique ayant le plus de trafic.

C – Performance sans prendre en considération l'aspect économique

Les stratégies suivantes prennent généralement en compte la performance SLA et ajustent de manière élastique le nombre de répliques. Certaines stratégies telles que RepliC [7] effectuent la réplication dans un environnement de base de données élastique à locataires multiples. RepliC surveille l'utilisation du système par rapport aux changements de la charge de travail afin de gérer la variation en dirigeant les transactions vers les répliques qui ont des ressources disponibles. Par conséquent, la stratégie RepliC satisfait la qualité de service (temps de réponse) avec des violations de SLA minimales. L'augmentation de la localité des données, plus précisément de la localité spatiale grâce à des rendements de réplication des données, permet d'améliorer les performances en plaçant stratégiquement les répliques plus près des sites demandeurs. Même si certaines stratégies ne considèrent pas directement les performances comme objectif, l'augmentation de la performance est observée à la suite de la réplication de données [97]. Une autre forme de localité croissante des données pour améliorer les performances se concentre sur la localité temporelle. De cette manière, certaines stratégies font valoir que les ensembles de données fréquemment consultés resteront probablement populaires à l'avenir [98] et décideront ce qu'il faut répliquer selon l'évaluation de la popularité mentionnée. C'est une stratégie simple mais efficace pour satisfaire les garanties de performance comme en témoigne la stratégie RTRM (Response Time-based Replica Management) [99]. Lorsque l'accès à des

données populaires entraîne un temps de réponse moyen supérieur à celui souhaité, ces ensembles de données sont répliqués pour résoudre le problème de performances. Ce qui est intéressant à propos de RTRM, c'est qu'il ne reproduit pas toutes les données, sauf les données populaires, afin de conserver l'utilisation des ressources.

L'idée de provisionnement dynamique des données est utilisée dans les systèmes de gestion de bases de données dans le Cloud. Sakr et Liu [100] ont introduit une stratégie centrée sur le client, orientée SLA. Ils ont présenté un framework qui permet de faciliter le provisionnement adaptatif et dynamique des bases de données des applications logicielles en fonction des politiques définies par l'application pour satisfaire leurs propres exigences de performance SLA, éviter les pénalités de toute violation SLA et contrôler le coût monétaire des ressources de calcul allouées. Le framework surveille en permanence la charge de travail des serveurs de base de données, suit la satisfaction du SLA défini par l'application et prend les mesures nécessaires en cas de besoin. Le framework est indépendant de la plateforme de base de données et repose sur un mécanisme de réplication de base de données basé sur la virtualisation.

Dans les réseaux de diffusion de contenu, l'objectif de performance peut consister simplement à fournir les données au demandeur. En ce sens, AREN (an Adaptive Replication scheme for Edge Networks) [16] est un bon exemple d'application de SLA en tant que garantie de livraison de données. AREN effectue une réplication de données prenant en compte la popularité pour optimiser l'utilisation de la bande passante afin de réduire les violations de SLA et la consommation de stockage ce qui améliore l'expérience globale de l'utilisateur.

Avec un objectif de performance similaire, Zhao [101] traite le délai de réplication, c'est-à-dire la rapidité avec laquelle un serveur de base de données actualise toutes les répliques. Si le délai de réplication est supérieur à un seuil prédéterminé, de nouvelles répliques sont créées pour résoudre la dégradation des performances. Cette stratégie vise à satisfaire le SLA de façon centrée sur le locataire. Dans une autre étude [102], les auteurs discutent un système de gestion de base de données élastique, basée sur le Cloud qui permet la gestion automatique des tâches de gestion des données, y compris une implémentation basée sur des règles de réplication.

Lin et al. [96] ont étudié le problème de la réplication des données sous contrainte

de QoS (QADR : QoS-Aware Data Replication) dans les systèmes de Cloud Computing. Les auteurs considèrent que l'exigence de QoS d'une application est déterminée à partir des informations de la requête, tel que le temps de réponse. Pour résoudre le problème QADR, deux algorithmes de réplication ont été proposés. Le premier algorithme appelé HQFR (High-QoS First-Replication). Dans cet algorithme, si une application a une exigence de qualité de service plus élevée, elle aura la priorité par rapport aux autres applications pour effectuer la réplication des données. Cependant, l'algorithme HQFR ne peut pas atteindre l'objectif souhaité. Pour trouver la solution optimale du problème QADR d'une manière efficace, ils ont proposé un nouvel algorithme. Dans cet algorithme, le problème QADR est transformé en problème MCMF (Minimum-Cost Maximum-Flow). Ensuite, un algorithme MCMF existant est utilisé pour résoudre de façon optimale le problème QADR en temps polynomial. Comparé à l'algorithme HQFR, l'algorithme optimal prend plus de temps de calcul. Les deux algorithmes de réplication proposés fonctionnent en temps polynomial. Leur complexité temporelle dépend du nombre de nœuds dans le système de Cloud Computing.

Agarwal et al. [103] ont proposé une approche peer-to-peer pour le partage de données dans les applications de workflow fonctionnant sur le Cloud. Ils ont décrit la conception et la mise en œuvre d'un gestionnaire de fichiers peer-to-peer qui utilise la mise en cache pour essayer d'améliorer les performances des applications de workflow à écriture unique dans des environnements distribués.

D – Performance avec la prise en considération de l'aspect économique

Afin de réduire les coûts de stockage et d'utiliser pleinement les ressources de stockage dans les systèmes Cloud. Des stratégies de réplication de données capables de gérer de façon adaptative les exigences de performance avec la prise en considération de l'aspect économique dans le Cloud sont nécessaires. Dans cette partie nous allons citer quelques stratégies de réplication qui traitent ces deux problèmes. Shen et al. [104] ont abordé le problème suivant : comment placer les répliques des images des machines virtuelles (VMs) pour équilibrer les charges de travail entre les nœuds de stockage ? Ils apportent principalement leur contribution sous trois aspects. Tout d'abord, ils ont procédé à l'étude de traçage et ont analysé les caractéristiques d'approvisionnement d'images de VM dans l'environnement de Cloud réel. Deuxiè-

mement, pour répondre aux problèmes de performance et de disponibilité causés par la popularité variée entre les différentes images, ils ont proposé un système de gestion de réplication d'image de VM rentable, dans lequel ils ont modélisé le système en tant que système de file d'attente et ont optimisé le nombre et le placement des répliques d'images VM. Troisièmement, ils ont implémenté leur système de gestion d'images à l'aide d'un logiciel de Cloud open-source populaire et l'ont déployé dans un centre de données pour évaluer son efficacité.

Dans un contexte différent, Zeng et al. [105] abordent le problème de placement de réplication dans les systèmes de stockage de Cloud. Ils ont proposé des placements de répliques sous contrainte du coût monétaire et de la QoS dans les systèmes de stockage de Cloud. Un modèle mathématique est construit pour minimiser le coût de réplication, et deux algorithmes sous contrainte du coût monétaire et de QoS sont proposés pour résoudre le problème. Dans le premier algorithme GS_QoS (Greedy Site Algorithm), un nœud ayant l'utilité la plus élevée est sélectionné pour la réplication en premier, puis des utilisateurs potentiels lui sont assignés dans chaque tour. La sélection est répétée jusqu'à ce que tous les utilisateurs soient affectés. Cet algorithme entraîne un gaspillage si un nœud sélectionné n'a pas un grand nombre d'utilisateurs potentiels. Une amélioration de l'algorithme GS_QoS a été proposée, nommée GS_QoS_C1 (Improved Greedy Site Algorithm with constraint). Dans GS_QoS_C1 une contrainte supplémentaire est utilisée pour déterminer s'il faut ou non sélectionner un nouveau nœud pour la réplication. Dans GS_QoS_C1, ils vérifient s'il y a suffisamment d'utilisateurs potentiels pour que le nœud soit sélectionné.

Zeng et al. [106] ont étudié la création et le placement de répliques dans des systèmes de stockage de Cloud en prenant en compte les exigences de QoS. Dans leur travail, la hiérarchie du Cloud est décrite comme des fournisseurs de services qui achètent ces services auprès de vendeurs de services Cloud et, à leur tour, des clients qui achètent des services auprès de ces fournisseurs de services. Les auteurs visent à minimiser les coûts des fournisseurs de services tout en maximisant l'utilisation du stockage pour les utilisateurs. Ils mesurent la qualité du service et calculent une mesure d'importance des ensembles de données. Ce qu'il faut répliquer est alors déterminé en fonction de la métrique d'importance. Le placement des répliques est effectué de manière à maximiser le volume de transfert de données par dépense

unitaire.

E – Stratégies de réplication de données avec autres objectifs

Dans la littérature nous trouvons d'autres stratégies de réplication de données qui visent à améliorer d'autres objectifs tel que : la latence [107], [108], la consommation d'énergie ou l'équilibrage de charge [109].

Une stratégie de création de répliques dynamiques basée sur l'accélération est présentée pour les Clouds de stockage [107]. Pour réduire la latence d'accès, la stratégie prédit les fichiers chauds (accès fréquent) en termes d'accélération en se basant sur le principe de localité d'accès. La stratégie proposée permet de prédire avec plus de précision la prochaine donnée chaude et donc elle a des meilleurs effets sur la réduction de la latence du réseau.

Sharov et al. [108] présentent une stratégie de réplication de données pour le stockage de Cloud basé sur les leaders. Une réplique de chaque ensemble de données est sélectionnée en tant que leader qui coordonne les tâches de réplication. Alors que toutes les répliques peuvent être accédées en lecture, les leaders peuvent être accédées en lecture ou en écriture. Ils proposent trois algorithmes pour gérer le placement des leaders, des votants et des répliques afin de réduire la latence des opérations d'accès aux données. Ils montrent que leur stratégie améliore la latence de l'accès aux données jusqu'à 50% dans un système de fichiers distribué.

Dans [110], les auteurs abordent le problème de la conservation de l'énergie pour les grands centres de données qui exécutent des travaux MapReduce. Ils ont constaté que les serveurs sont plus efficaces lorsqu'ils sont utilisés à un niveau d'utilisation plus élevé. Dans ce contexte, ils ont proposé un algorithme de placement de données et de reconfiguration de cluster qui met à l'échelle dynamiquement le cluster en fonction de la charge de travail qui lui est imposée. Cependant, l'efficacité de cette approche dépend de la vitesse à laquelle un nœud peut être activé. Les résultats de simulation montrent des économies d'énergie allant jusqu'à 54% pour des charges de travail faibles et environ 33% pour des charges de travail moyennes.

Dans le travail [109], les auteurs proposent d'utiliser des clusters dans un Cloud, de nombreux ordinateurs ou nœuds de stockage sont connectés ensemble pour fournir

un serveur de stockage de haute capacité. Les nœuds de stockage sont connectés dans un cluster, ce qui peut entraîner des problèmes évidents d'équilibrage de charge et des problèmes d'évolutivité. Ils proposent un système de gestion de réplication pour choisir le nombre de répliques optimal. Afin d'atteindre l'équilibre de charge du cluster, la pondération des nœuds est appliquée sur le cluster. Le placement de répliques proposé utilisant la pondération des nœuds rend les charges de stockage des nœuds plus équilibrées lorsque des blocs de données sont ajoutés.

F – Stratégies de réplication de données multi-objectifs

Les stratégies de réplication de données basées sur un modèle d'optimisation multi-objectifs [83] sont des stratégies intéressantes, il est possible de satisfaire plus d'un SLO simultanément tel que la disponibilité, le temps de réponse, la latence du réseau ou la fiabilité. Ces modèles mathématiques peuvent être effectués par une fonction d'optimisation qui prend en compte plusieurs objectifs.

Liao et al. [111] ont formulé un modèle mathématique pour décrire la relation entre les exigences de QoS et le nombre de répliques. Ils ont proposé une stratégie dynamique de suppression des répliques de données sous contrainte de QoS nommée DRDS (Data Replicas Delete Strategy). Cette stratégie peut économiser de l'espace disque, de réduire le coût de maintenance et de garantir la qualité de service exigée.

Sun et al. [17] ont présenté une stratégie de réplication de données dynamique pour un système de Cloud hiérarchique à plusieurs niveaux. Il s'efforce d'augmenter la disponibilité des données, d'améliorer le taux d'exécution des tâches et de minimiser la consommation de bande passante du système Cloud. Leurs contributions peuvent être résumées comme suit :

- Un modèle mathématique est formulé pour décrire la relation entre la disponibilité du système et le nombre de répliques ;
- Les données populaires sont identifiées et l'opération de réplication correspondante est déclenchée lorsque la popularité d'un fichier de données dépasse un seuil dynamique ;
- Les répliques sont placées dans les nœuds de stockage de manière équilibrée.

Wei et al. [18] ont proposé une stratégie de réplication dynamique rentable appe-

lée CDRM (Cost-effective Dynamic Replication Management). Ils soulignent que le fait d'avoir trop de répliques n'augmente pas la disponibilité mais entraîne des rendements réduits. Le CDRM calcule et maintient un nombre minimum de répliques pour satisfaire un niveau de disponibilité donné. Dans le stockage de Cloud considéré, les ensembles de données sont fragmentés en plusieurs blocs. Les auteurs calculent la disponibilité des ensembles de données en fonction de la disponibilité de chacun de leurs fragments. Avec ces informations, CDRM calcule un nombre de répliques requis pour chaque fragment. Le placement des répliques est effectué de manière à équilibrer la charge sur tous les sites. La réduction des accès inefficaces garantit que tous les fragments sont servis sans provoquer de goulot d'étranglement. Les auteurs observent également des performances d'accès accrues.

Un objectif intéressant de la réplication des données est de cibler spécifiquement certaines dépenses particulièrement intéressantes dans le Cloud. À cet égard, certaines stratégies de réplication des données [112], [113] se concentrent sur l'efficacité énergétique et les performances des applications Cloud. Boru et al. [112] proposent un algorithme de réplication de données qui prend en compte la consommation d'énergie et la bande passante requises pour l'accès aux données. Chaque objet de données est disponible dans la base de données centrale et, en fonction des observations historiques de la fréquence d'accès, les données peuvent être répliquées dans la base de données au niveau Datacenter et dans les bases de données au niveau serveurs. Un module appelé RM (Replica Manager) situé à la base de données centrale analyse périodiquement les historiques d'accès afin d'identifier les objets de données qui doivent être répliqués et l'emplacement des nouvelles répliques. RM calcule les taux d'accès et de mise à jour dans les intervalles précédents et effectue une estimation de leurs valeurs futures. Selon cette estimation qui inclus l'énergie et la demande de la bande passante, un placement approprié est trouvé pour les répliques afin de minimiser leur consommation d'énergie et de bande passante dans les intervalles à venir.

Nous allons résumer dans les tableaux 1.1 et 1.2 les différentes stratégies de réplications de données citées dans cette section. Pour chaque stratégie, nous précisons l'objectif principal défini par les auteurs, le simulateur utilisé pour évaluer la stratégie proposée ainsi les paramètres de simulation.

TAB. 1.1 – Comparaisons entre les différentes stratégies de réplication de données dans les Clouds

Ref	**Objectif(s)**	**Aspect Economique**	**Simulateur**	**Paramètres de simulation**
[7]	Performance		OLTP Benchmark	temps de réponse, violations de SLA
[16]	Performance		PeerSim	utilisation de la bande passante, durée de récupération
[17]	Disponibilité, Performance		CloudSim	taux d'exécution réussi, temps de réponse
[18]	Disponibilité, Equilibrage de charge		Environnement réel : Hadoop	latence moyenne, l'utilisation de stockage, nombre de répliques, disponibilité
[84]	Disponibilité	X	CloudSim	Cout de réplication, Disponibilité
[88]	Fiabilité	X	N'est pas mentionné	nombre de répliques moyen, prix de service
[90]	Disponibilité		CloudSim	temps de réponse, nombre de répliques
[91]	Disponibilité		N'est pas mentionné	nombre de blocks redondants, disponibilité
[92]	Disponibilité	X	CloudSim	temps de réponse, equilibrage de charge, bande passante
[93]	Fiabilité	X	Environnement de Cloud réel	temps d'analyse des données, temps de traitement, coûts de stockage
[94]	Fiabilité		Environnement de Cloud réel	performance d'accès à la donnée, durabilité de stockage, surcoût d'exécution,
[95]	Fiabilité		N'est pas mentionné	equilibrage de charge, nombre de nœuds en panne, and récupération, coût de migration, coût de réplication

TAB. 1.2 – Comparaisons entre les différentes stratégies de réplication de données dans les Clouds

Ref	**Objectif(s)**	**Aspect Economique**	**Simulateur**	**Paramètres de simulation**
[96]	Performance		MatLab	temps d'exécution, coût de réplication, le temps moyen de récupération,
[97]	Performance		Environnement réel, Hadoop	temps d'execution, le nombre de tâches
[98]	Performance		Non évalué	
[99]	Performance		OptorSim	temps d'exécution moyen, l'utilisation du réseau
[100]	Performance		berkeley Cloudstone	satisfaction de SLA
[101]	Performance		Environnement réel	délai de réplication, temps de réponses
[103]	Performance		Environnement de Cloud rééel : Amazon's	temps de réponse moyen, débit
[104]	Performance	X	Environnement réel : SEUCloud	disponibilité, le taux de provision
[105]	Performance	X	Leur propre simulateur	coût relatif, performance CDF
[106]	Performance	X	Leur propre simulateur	délai total, le taux d'acceptation, performance CDF, coût relative
[107]	Latence		Leur propre simulateur	temps de transmission
[108]	Latence		Leurs propre outils	% de réduction de la latence, % de surcoût de la latence
[109]	Equilibrage de charge		Environnement réel : Hadoop	utilisation de stockage, équilibrage de charge
[110]	Energie		GridSim	énergie économisée, nombre de nœuds actifs
[112] [113]	Performance, Energie		GreenCloud	énergie, bande passante, le délai d'accès à la donnée

1.4 Conclusion

Dans ce chapitre, nous avons étudié plusieurs stratégies de réplication dans les systèmes à grande échelles. Plusieurs études ont essayé de classer les stratégies de réplications dans les grilles de calculs. Parmi les classifications existantes, on trouve les stratégies de réplication statiques vs dynamique, ou des stratégies de réplications centralisées vs décentralisées ; d'autres auteurs ont proposé d'autres classification selon la fonction objective ou selon l'architecture de la grille. Dans le Cloud Computing, peu d'études ont été proposé pour classer les stratégies de réplication, Milani et al. [13] ont proposé une classification en deux groupes principaux, statique et dynamique. Nous avons présenté une classification des stratégies de réplication de données selon des fonctions objectives telle que : la disponibilité, la fiabilité, la performance et des stratégies de réplication qui visent à améliorer plusieurs objectifs en même temps.

CHAPITRE 2

Stratégie proposée pour gérer la réplication de données

2.1 Introduction

Il existe un certain nombre de travaux dans la littérature qui traitent la réplication des données dans les systèmes de Cloud. Des stratégies ont été proposées pour améliorer les performances, réduire la consommation de bande passante, augmenter le niveau de disponibilité des données et assurer l'équilibrage des charges. Cependant, ces objectifs semblent être contradictoires. Par exemple, la réplication des données garantit la disponibilité. Mais, cela se fait au détriment des communications entre les sites qui surchargent le réseau et donc un effet sur les performances. De plus, la plupart de ces stratégies négligent le coût de réplication et ne prennent pas en compte le profit du fournisseur.

Dans ce chapitre nous présentons notre stratégie de réplication de données dynamique qui répond simultanément aux exigences de disponibilité et de performance tout en prenant en compte le profit du fournisseur. Nous décrivons la topologie du Cloud puis nous détaillons notre stratégie de réplication nomée DRAPP (Data Replication strategy that delivers Availability and Performance requirements while ensuring Profit of Cloud Providers).

2.2 Topologie de Cloud

Les environnements de Cloud Computing ont principalement une topologie de structure hiérarchique. Les fournisseurs de Cloud rendent une topologie hiérarchique accessible aux utilisateurs du Cloud afin de tirer parti des besoins spécifiques des applications et des services, Nous considérons une topologie de Cloud hiérarchique qui prend en charge la réplication de données. Les fournisseurs de Cloud établissent souvent de multiples installations dans des régions géographiques distinctes pour une multitude raisons, y compris la fourniture de services dans le monde entier. Chaque région peut contenir plusieurs centres de données répartis. Ces centres de données sont des installations de Cloud qui hébergent un certain nombre de machines virtuelles qui fournissent une puissance de calcul, des capacités de bande passante du réseau et un stockage aux locataires. Tout au long de ce document, nous nous référons à une machine virtuelle par un nœud. Tous les nœuds du Cloud sont interconnectés par des liens réseau. Les nœuds dans le même centre de données d'une région géographique particulière sont interconnectés via des liaisons locales moins chères. De la même manière, la bande passante intra-région est relativement faible et plus chère.

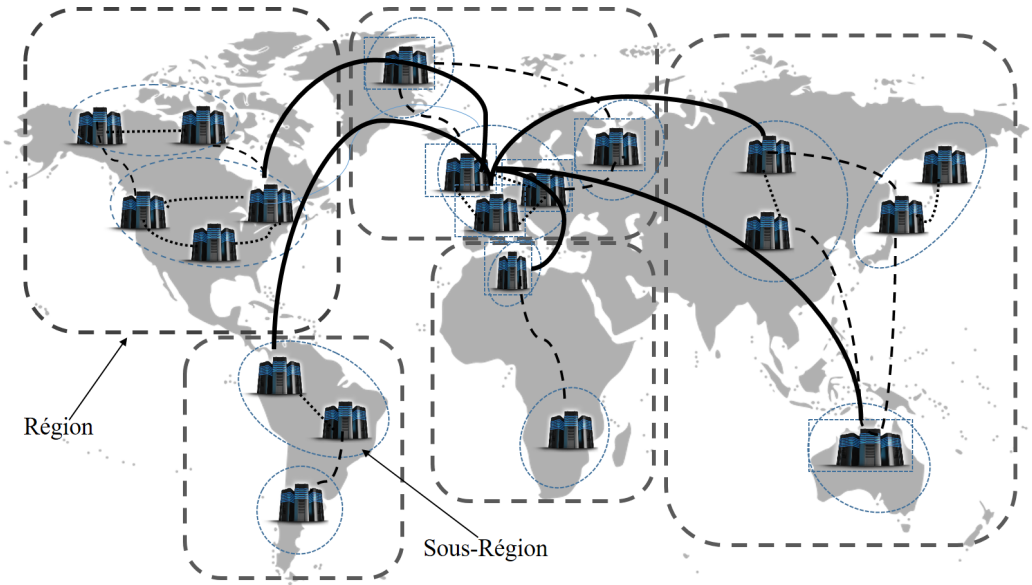


FIG. 2.1 – Topologie de Cloud avec les régions et les sous-régions.

2.3 Schéma de réplication

Les problèmes majeurs de la réplication de données comprennent trois défis importants qui doivent être résolus :

i. Quelles données doivent être répliquées et quand répliquer pour répondre à la fois aux exigences des utilisateurs et au bénéfice économique du fournisseur ? par exemple, des données répliquées trop tôt ne réduiront pas le temps d'attente ou n'accéléreront pas l'accès aux données.

ii. Combien de nouvelles répliques appropriées devraient être créées pour répondre à une exigence raisonnable de disponibilité du système ? À mesure que le nombre de répliques augmentent, la disponibilité des données augmentent aussi. Mais en même temps, les coûts de création et de maintien des répliques vont augmenter de manière significative ce qui engendre des dépenses inutiles. En plus, le fournisseur devrait avoir un réel profit économique.

iii. Où les nouvelles répliques doivent être placées pour répondre aux exigences du temps d'exécution des tâches et de la consommation de la bande passante ?

Dans le Cloud Computing, toutes les ressources de calcul (comme le stockage, les données) sont partagées entre les utilisateurs. En conséquence, une gestion élastique de ces ressources permet de satisfaire à la fois les exigences des utilisateurs et le profit du fournisseur. L'évaluation de la qualité du service (QoS) délivrée dans le Cloud Computing est extrêmement importante. Pour ce faire, un contrat de niveau de service (SLA), un contrat légal entre le locataire et le fournisseur, est signé. Principalement, il comprend : a) un montant payé par le locataire au fournisseur pour le traitement de ses demandes ; b) un ou plusieurs objectifs de niveau de service SLO qui sont les exigences du locataire qui doivent être satisfaites par le fournisseur (ex : disponibilité, temps de réponse,...). Par exemple le temps de réponse maximum (un seuil) qui peut être fourni par le fournisseur lors de l'exécution d'une requête d'un locataire. c) une période de validité du contrat, d) Une pénalité payée par le fournisseur à son locataire en cas de violation de SLA.

Notre stratégie DRAPP peut être intégrée à un framework afin de créer un module de réplication de données comme indiqué dans la Figure 2.2. Le module de réplication

est composé de trois agents : agent de surveillance, agent de traitement et agent de déclenchement de réplication.

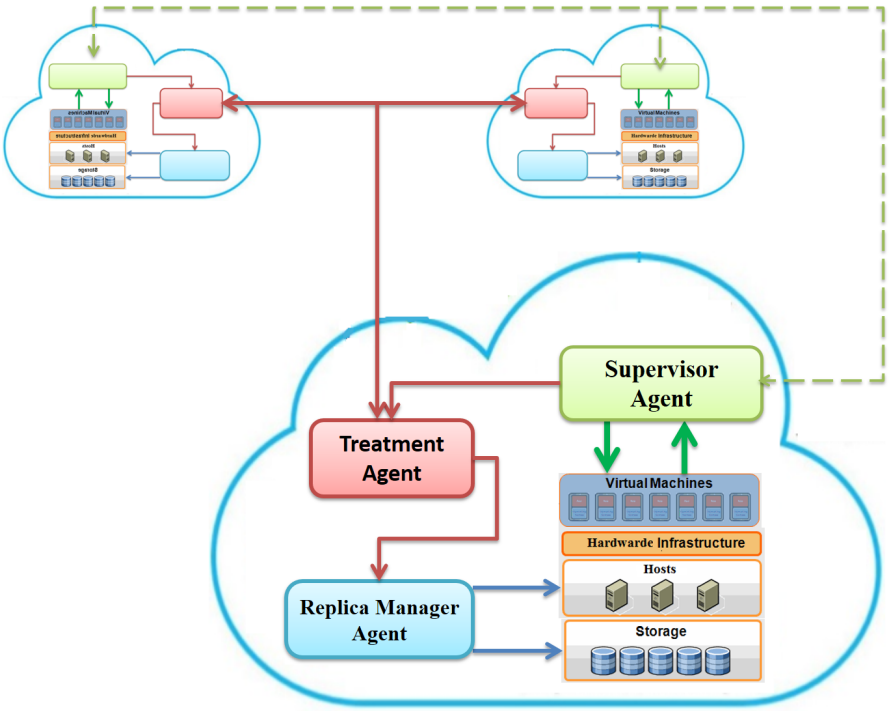


FIG. 2.2 – Architecture de la stratégie DRAPP

A partir de cette architecture que nous décrivons dans les sous-sections suivantes, nous distinguons trois phases principales : (i) la surveillance, (ii) le traitement et (ii) le déclenchement de réplication comme indiqué sur la Figure 2.3.

2.3.1 Phase de surveillance

Cette phase correspond à la supervision du Cloud. Elle consiste à collecter des informations sur les ressources existantes. Un exemple de cette information est la fréquence de lecture des données qui est utilisée pour calculer la popularité de chaque ensemble de données. Un autre exemple est la charge de chaque machine virtuelle dans chaque centre de données. Ce paramètre est important puisque les nouvelles répliques doivent être créées sur des nœuds sous-chargés.

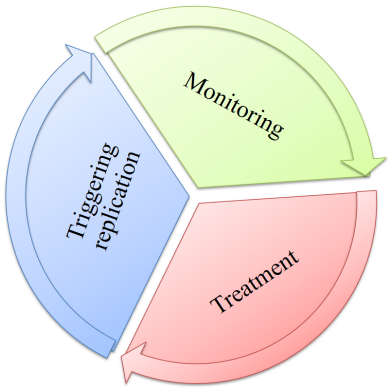


FIG. 2.3 – Phases opérationnelles

2.3.2 Phase de traitement

La réplication dynamique est basée sur les historiques d'accès au fichier. Elle consiste à analyser les informations produites par la phase de surveillance afin d'identifier l'état du système. Notre stratégie est basée sur un modèle de coût qui vise à estimer le temps de réponse de toute requête avant son exécution. Nous basons sur le travail [11] pour calculer l'estimation du temps de réponse de chaque requête. Si l'estimation du temps de réponse est supérieure au seuil défini dans le SLA, la réplication est envisagée afin d'éviter les pénalités. Le bénéfice du fournisseur doit également être estimé afin de ne répliquer que si la réplication génère un bénéfice économique réel pour le fournisseur. Cette phase traite également la suppression des répliques. Si une réplique est inutile, elle doit être supprimée afin de réduire les dépenses du fournisseur. En conséquence, son profit économique est augmenté.

2.3.3 Phase de déclenchement de réplication

Cette phase est basée sur les résultats de la phase précédente. La Figure 2.4 met en évidence les principales étapes pour traiter les requêtes des locataires. Lorsqu'une réplication est envisagée, l'étape suivante consiste à calculer le nombre de répliques pour chaque donnée demandée dans chaque région. Ensuite, nous prenons en considération le budget de chaque sous-région pour déterminer le nombre adéquat à créer dans cette sous-région. Ces répliques sont placées de manière équilibrée afin que la réplication soit rentable et elle permet de générer un profit réel pour le fournisseur.

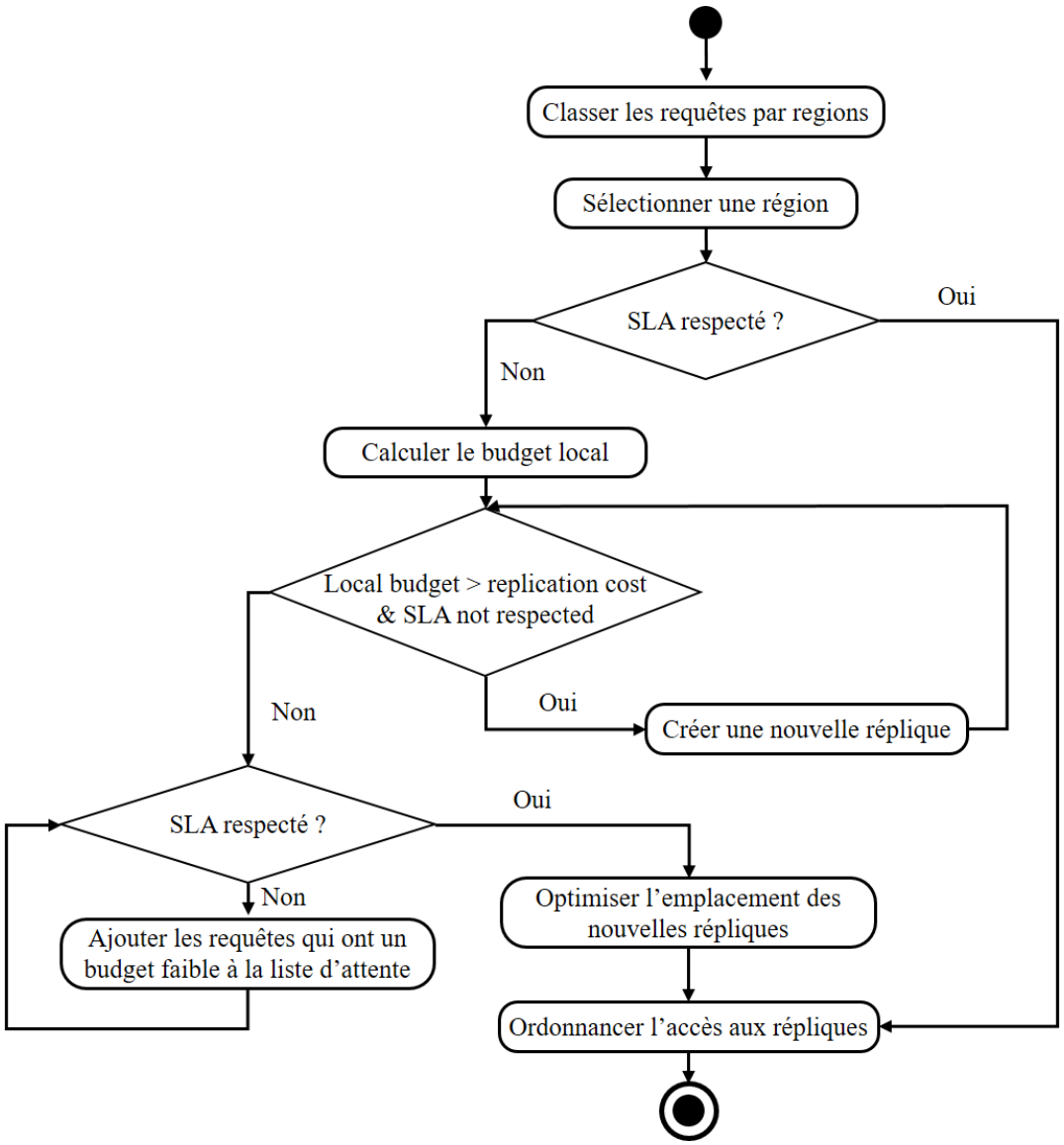


FIG. 2.4 – Principales étapes de DRAPP

Dans le tableau suivant, nous décrivons le rôle de chaque agent dans les trois phases du DRAPP.

Elément			
Phase fonctionnelle	**Agent surveillant**	**Agent de traitement**	**Agent de gestion de répliques**
Surveillance	- Collecter des informations (fréquence de lecture des données, budget, etc.)		
Traitement	- Contrôler l'équilibrage de charge de chaque machine virtuelle	- Estimer le temps de réponse - Planification de l'accès aux répliques	
Déclenchement de réplication	- Identifier les fichiers à répliquer	- Déterminer quand démarrer le processus de réplication - Décider si une réplique inutile doit être supprimée	- Calculer le nombre minimum de répliques requises - Choisir le meilleur emplacement pour les nouvelles répliques

TAB. 2.1 – Rôle de chaque agent dans les différentes phases du DRAPP

2.4 Stratégie de réplication proposée

Une solution idéale pour améliorer la disponibilité des données consiste à répliquer les données dans tous les nœuds. Ainsi, le coût d'accès aux données sera considérablement réduit. Cependant, cette solution n'est pas réaliste en raison des contraintes de stockage et de bande passante. Donc, des stratégies de réplication de données sont nécessaires.

Une stratégie de réplication doit traiter les problèmes suivants : (i) Quelles données ont besoin d'être répliquées ? (ii) quand répliquer ? (iii) où placer les répliques ? et (iv) combien de répliques sont-elles conservées pour répondre aux besoins des utilisateurs et aux avantages économiques du fournisseur ? Les objectifs possibles d'une stratégie de réplication sont d'exploiter la popularité en répliquant les ensembles

de données les plus demandés [114], de minimiser le coût de mise à jour [7] ou de maximiser les objectifs économiques [100].

Cette section traite les quatre problèmes présentés ci-dessus. L'idée générale est de créer de nouvelles répliques en fonction des exigences de disponibilité et de performance SLO. En plus, nous prenons en considération le bénéfice du fournisseur. Les requêtes des locataires sont classées par régions. Les ressources disponibles sont analysées afin de vérifier si elles répondent aux exigences des requêtes de locataire. Cela permet d'éviter de créer de nouvelles répliques inutiles.

2.4.1 Quand et quoi répliquer ?

Une condition importante pour déclencher une réplication est la violation de l'objectif de temps de réponse. La réplication est déclenchée lorsque l'estimation du temps de réponse des requêtes dépasse le temps de réponse convenu dans le SLA. Nous avons utilisé l'estimation du temps de réponse proposée dans [11], [41], [82]. Les informations statistiques sont collectées, dans la phase de surveillance, afin de les utiliser lors du déclenchement de processus de réplication. Un enregistreur d'accès est attribué à chaque hôte, qui est utilisée pour stocker l'enregistrement d'accès des locataires aux répliques, y compris le nom du fichier, le nombre d'accès, la taille du fichier, etc. L'agent surveillant examine les fichiers demandés par les locataires. Seulement les données requises par des requêtes des locateurs qui génèrent une violation de SLA sont concernées par le processus de réplication. D'un autre côté, les répliques ne sont créées que si le bénéfice du fournisseur dépasse les coûts de réplication.

2.4.2 Estimation du bénéfice du fournisseur

Une nouvelle réplique est créée lors de l'exécution d'une requête seulement si le fournisseur a un réel bénéfice. Cela signifie que ses revenus sont supérieurs à ses dépenses. Dans ce qui suit, nous estimons à la fois les revenus et les dépenses du fournisseur.

A – Les revenus du fournisseur

Lors de l'exécution d'une requête Q d'un client, le fournisseur reçoit un loyer de ce client. Le loyer dépend de différents paramètres tels que la durée d'utilisation et

les objectifs à satisfaire. Dans ce contexte, nous nous concentrons sur les objectifs de disponibilité et de performance. Le locataire n'est pas facturé lorsque le fournisseur déclenche le processus de réplication de données afin de satisfaire les exigences SLO. En d'autres termes, la réplication de données est transparente pour le locataire. Par conséquent, il est évident que l'augmentation du nombre de locataires diminue la performance par locataire. Cela réduit le coût d'exploitation global du fournisseur, ce qui augmente ses revenus. En conséquence, son bénéfice est augmenté [55].

B – Dépenses du fournisseur

Le fournisseur a un certain nombre de dépenses lors de l'exécution d'une requête Q comme il est montré dans l'équation 2.1. Cela inclut le coût d'utilisation de toutes les ressources nécessaires pour exécuter la requête Q. Il inclut le coût d'utilisation du CPU pour exécuter les sous-requêtes parallèles qui constituent Q. Soit TT_Q le temps total estimé nécessaire pour exécuter une requête Q. Soit Nb le nombre de nœuds requis pour exécuter Q pendant une unité de temps U_T, par exemple une heure sur un nœud avec Amazon. Ces nœuds incluent également ceux qui contiennent les répliques créées lors de l'exécution de Q [11].

$$Dépenses = C_N + C_S + C_I + C_P + \sum_{1}^{Nb}(TT_Q \times U_T) \tag{2.1}$$

Les dépenses du fournisseur contiennent également le coût d'utilisation du réseau (C_N) pour envoyer les données distantes à partir d'autres nœuds. Le coût du réseau inclut également le coût de la migration des données ou le placement de nouvelles répliques sur des nœuds distants, ce qui nécessite des ressources de bande passante supplémentaire. Le coût de stockage (C_S) est un autre coût que le fournisseur doit prendre en compte lors de la création de nouvelles répliques. Nous traitons également le coût d'investissement (C_I) qui consiste à payer le logiciel nécessaire pour l'exécution des requêtes des locataires. Le coût C_P est un autre coût important qui consiste au paiement de pénalités probables aux locataires lorsqu'un ou plusieurs objectifs SLO ne sont pas atteints. Une pénalité est payée par le fournisseur au locataire lorsqu'une violation de SLA survient. Dans le DRAPP, si le temps de réponse est supérieur au seuil de temps de réponse, le fournisseur doit payer une pénalité au locataire.

2.4.3 Combien de répliques ?

Connaître le nombre approprié des nouvelles répliques qui doivent être créées dans le Cloud est important pour répondre à une exigence raisonnable de disponibilité du système. Déterminer ce nombre de répliques est une tâche extrêmement difficile dans le Cloud, en particulier avec l'augmentation énorme et rapide des nœuds et des ressources. En plus, avec l'augmentation du nombre de nouvelles répliques, le coût de maintenance du système augmentera considérablement. Cependant, un trop grand nombre de répliques n'augmente pas la disponibilité, mais entraîne plutôt des dépenses inutiles. Pour résoudre ces problèmes, nous avons proposé un algorithme optimisé de création de répliques. L'algorithme 1 répond à la question : combien de copies doivent être créées ? Notre algorithme est basé sur un modèle de coût afin de profiter au maximum des avantages de la réplication. Il calcule le nombre minimum de répliques à créer pour maintenir une haute disponibilité des données pendant que les performances du système sont améliorées.

Il est préférable de traiter les requêtes des locataires dans les centres de données proches de leurs régions. Pour cela, nous assignons les requêtes aux régions appropriées. Ensuite, nous estimons le nombre des répliques de données nécessaires pour l'exécution de ces requêtes. Pour déterminer ce nombre de répliques, nous créons d'une façon itérative une nouvelle réplique tant que le SLA n'est pas satisfait. Au même temps, le fournisseur devrait avoir un bénéfice. La création de nouvelle réplique s'arrête si les exigences énoncées dans le SLA sont satisfaites ou si la marge bénéficiaire est faible.

La disponibilité est une exigence critique pour de nombreuses applications de données. Elle peut avoir un impact important sur le coût financier d'un service donné. Un nombre de répliques insuffisant peut être trop coûteux en introduisant un surcoût élevé sur la performance et en réduisant la disponibilité des données. Cela a un impact négatif sur les bénéfices économiques des fournisseurs de service, il génère des pertes financières importantes car le SLA est violée.

L'algorithme proposé optimise les bénéfices du fournisseur et garantit une QoS. Lorsqu'une requête de client est soumise, il prend en compte deux options :

– La première option est de créer de nouvelles répliques pour satisfaire les objec-

tives SLO (disponibilité et performance), ce qui évite le paiement des pénalités causées par la violation du SLA. La création de nouvelles répliques est effectuée uniquement si le profit minimum du fournisseur est assuré,

```
Classify requests by regions;
forall region R_i do
    Estimate the average response time TRM_i for the region R_i ;
    if the time TRM_i respects the SLA then
        Scheduling access to replicas;
    else
        Calculate the local budget;
        Bl_Ri = sum_{K in R_i} BudReq_k ;
        while Bl_Ri > cost_of_replication and SLA not respected do
            NumberRep_Ri++; /* Increment the number of replicas */
            Check SLA; /* check the quality of services with the
               new replicas                                      */
        end
        if SLA is respected then
            Scheduling access to replicas;
        else
            // Here necessarily Bl_Ri < cost_of_replication
            while SLA is not respected do
                Remove requests that have a low budget; /* do not execute
                   them immediately to unload the nodes          */
                Add these requests to the waiting list;
                Check SLA;
            end
        end
    end
end
```

Algorithm 1: Déterminer le nombre minimum de répliques

– Dans le cas où le coût de réplication est élevé et ne garantit pas le profit minimum du fournisseur, la deuxième option consiste à placer les requêtes qui ont un faible budget dans une file d'attente. Les requêtes qui ont un budget élevé sont traitées en premier. Le traitement des requêtes qui ont un faible budget est lancé après la libération des répliques. Si les SLO de ces requêtes ne sont pas satisfaits, Un paiement de pénalités est considéré. Par conséquent, payer certaines pénalités tout en recevant des revenus élevés des autres locataires augmente le profit du fournisseur.

D'autre part, lorsque le temps de réponse estimé d'une requête donnée est beaucoup plus petit que le seuil de temps de réponse, nous pourrions supprimer certaines répliques qui ne sont pas demandées lors de la satisfaction du temps de réponse SLO. En conséquence, notre algorithme ajuste dynamiquement le nombre des répliques tout en satisfaisant les exigences SLO des locataires. Cela permet également aux fournisseurs de maximiser leurs gains en libérant les ressources qui ne sont pas utilisées.

2.4.4 Ordonnancement d'accès

Pour obtenir les meilleures performances, l'ordonnancement des requêtes et la réplication des données ont été combinés. Une bonne stratégie d'ordonnancement réduira le temps d'accès aux données et réduira aussi le temps global d'exécution des tâches dans le Cloud. Dans cette phase, le gestionnaire de traitement permet d'ordonnancer l'accès aux répliques (voir algorithme 2). Il sélectionne la meilleure réplique à la requête de l'utilisateur qui paie le plus. Cette pratique permet de réduire le temps d'accès et également de satisfaire les exigences maximales de cette requête. Pour planifier l'accès d'une requête Q à une réplique, nous devons trier les différentes requêtes par ordre décroissant en fonction de leur budget. La requête qui a le budget maximum $Req_{\max}$ est sélectionnée et la meilleure réplique disponible Rep_{best} est choisie pour cette requête.

2.4.5 Placement des répliques

Le placement des répliques consiste à trouver d'une manière efficace les emplacements physiques pour les différentes répliques que nous souhaitons créer dans le

Classify requests by regions;
foreach *request* $Req \in$ *the list of request of region* R_i **do**
    Sort queries in descending order according to the budget;
    Select the request Req_{max} that has the maximum budget;
    Select the best replica Rep_{best} for this request and execute the request;
    Remove Req from the list of requests
end

Algorithm 2: Ordonnancement d'accès à la réplique

système Cloud. La consommation de la bande passante constitue un problème important lors du placement des répliques. Le gestionnaire de répliques décide où les nouvelles répliques doivent être placées afin d'obtenir une meilleure disponibilité du système. Si les répliques et les requêtes sont distribuées de manière optimisée, les répliques peuvent améliorer l'accélération de l'accès aux données, réduire le temps d'attente et également diminuer la consommation de bande passante.

Après le calcul du nombre de répliques minimum pour chaque région en utilisant l'algorithme 1 et pour éviter de placer les nouvelles répliques d'une manière aléatoire, nous avons proposé l'algorithme 3 qui optimise le placement de ces nouvelles répliques. Dans un premier temps, nous distribuons le nombre des répliques aux différentes sous-régions en fonction de leurs budgets. Ensuite, nous créons plus de répliques dans les Sous-Régions SR qui ont un budget important pour satisfaire les exigences des locataires. Les locataires de certaines sous-régions bénéficient d'un accès meilleur aux répliques car ces locataires ont payé plus. Donc, ils peuvent accéder à des répliques dans leurs sous-régions ce qui permet d'améliorer le temps de réponse des requêtes de ces locataires.

Pour chaque région R_i, les différentes requêtes seront classées en fonction de leur sous-région SR. Ensuite, nous calculons le nombre de répliques dans chaque sous-région. Pour estimer le nombre de répliques dans une sous-région SR_j, nous devons calculer le budget local Bl_{SR_j} pour cette sous-région SR_j. Ensuite, nous déduisons le nombre de répliques $NombRep_{SRj}$ du nombre total de répliques $NombreRep_{Ri}$ dans

Classify requests by regions;
foreach *region* R_i **do**
  Classify requests according to the different sub-region SR;
  foreach *sub-region* SR_j **do**
    Calculate the local budget of sub-region Bl_{SR_j};
    Estimate the number of replicates for each sub-region SR_j;
    $NumbRep_{SR_j} = \left(NumberRep_{R_i} \times Bl_{SR_j}\right) / Bl_{R_i}$;
  end
  Make an approximation such that :

$$NumbRep_{SR_j} \text{ is an integer}$$
$$\&\&$$
$$\sum_{SR_j \in R_i} NumbRep_{SR_j} = NumbRep_{R_i}$$

  foreach *sub-region* SR_j **do**
    Create $NumbRep_{SR_j}$ replicas;
  end
end

Algorithm 3: Optimisation du placement des nouvelles répliques

la région Ri, le budget total Bl_{R_i} dans la région R_i et aussi le budget local Bl_{SR_j} de cette sous-région SR_j sont utilisés pour calculer le $NombRep_{SRj}$ comme indiqué dans l'équation suivante : $NumbRep_{SR_j} = \left(NumberRep_{R_i} \times Bl_{SR_j}\right) / Bl_{R_i}$. Nous faisons une approximation des nombres de répliques telles que la somme du nombre de répliques dans toutes les sous-régions est égale au nombre total de répliques dans la région R_i comme mentionné dans l'équation :

$$NumbRep_{SR_j} \text{ is an integer}$$
$$\&\& \tag{2.2}$$
$$\sum_{SR_j \in R_i} NumbRep_{SR_j} = NumbRep_{R_i}$$

Enfin, ces répliques sont placées dans les hôtes les plus performants dans les différents centres de donnée de ces sous-régions.

2.4.6 Suppression des répliques

Une stratégie de réplication de données dynamique doit permettre de créer et de supprimer automatiquement des répliques. La création et la suppression des répliques peuvent être réalisées en fonction de la variation des fréquences d'accès. Notre solution de suppression des répliques prend en charge la qualité de service, elle permet de réduire l'utilisation de l'espace disque et de minimiser les coûts de stockage et de maintenance. Au fil du temps, l'accès à certaines répliques peut diminuer en raison de l'évolution de la demande causée par les requêtes des locataires. Lorsqu'une réplique n'est plus nécessaire, elle sera supprimée. Pour déterminer quelles sont les répliques qui doivent être retirées du Cloud, nous utilisons un compteur pour calculer le nombre de visites pour chaque réplique. Les répliques qui ont la plus faible valeur seront considérées comme des répliques non performantes et seront supprimées. Le nombre de répliques maintenu après la suppression des répliques non nécessaire doit garantir les performances convenues dans le SLA.

2.5 Conclusion

Nous avons présenté notre stratégie DRAPP qui répond simultanément aux exigences de disponibilité et de performance tout en prenant en compte le profit du fournisseur. Notre idée générale est de créer de nouvelles répliques en fonction des contraintes de disponibilité et de performance (SLO). La création de nouvelles répliques est déclenchée uniquement si la contrainte de disponibilité des données n'est pas garantie ou si le temps de réponse d'une requête d'un client est supérieur au seuil de temps de réponse convenu dans le SLA. Certaines informations collectées, par exemple la charge de travail d'un nœud et la popularité des données, sont également utilisées lors de la prise de décision. Dans notre approche nous calculons le nombre de répliques minimum à créer afin de garantir que la QoS délivrée satisfait les attentes des locataires. Pour améliorer les performances, nous avons combiné la réplication et l'ordonnancement des requêtes. Ensuite, nous avons optimisé le placement des nouvelles répliques qui sont distribuées en fonction du budget. Enfin, nous avons

proposé une solution de suppression qui ajuste le nombre des répliques dynamiquement. Dans le chapitre suivant nous effectuons plusieurs séries d'expérimentations pour évaluer DRAP.

CHAPITRE 3

Evaluation expérimentale de la stratégie DRAPP

3.1 Introduction

Pour tester les nouvelles stratégies et techniques développées par les chercheurs, ces derniers ont besoin d'outils leur permettant d'évaluer ces nouvelles stratégies avant un déploiement réel dans un environnement où l'on peut reproduire les tests. Avec les Clouds, l'accès aux infrastructures nécessite de payer, même s'il s'agit juste de tester les besoins d'une application. Les outils de simulation offrent des avantages significatifs car ils permettent aux développeurs Cloud de tester gratuitement la performance, l'efficacité et la fiabilité de leurs stratégies dans un environnement reproductible et contrôlable. Ils offrent également la possibilité de changer les paramètres afin d'ajuster et optimiser ces stratégies avant un déploiement réel dans le Cloud. Ce chapitre est consacré à la réalisation et l'évaluation de notre stratégie de réplication de donnée DRAPP. Dans un premier temps, nous présentons l'environnement de notre travail, ensuite nous découvrons le simulateur CloudSim [19], [47], et finalement nous présentons une série de simulations et leurs interprétations pour mettre en évidence notre proposition.

3.2 CloudSim

L'objectif principal de simulateur CloudSim est de fournir un framework de simulation généralisé et extensible qui permet la modélisation, la simulation et l'expérimentation des nouvelles infrastructures du Cloud Computing et les services d'application, permettant aux utilisateurs de se concentrer sur des questions de conception du système qu'ils veulent étudier, sans être préoccupé aux détails relatifs aux services et infrastructures Cloud. Nous avons développé le simulateur sur une machine avec un processeur Intel Core i7 3537U / 2 GHz, doté d'une capacité mémoire de 8GB, sous Windows 10 de 64 bits. Nous avons utilisé l'environnement de développement Eclipse.

Architecture de CloudSim

La structure logicielle de CloudSim et ses composants est représentée par une architecture en couches comme il est montré par la Figure 3.1. Les premières versions de CloudSim utilise SimJava, un moteur de simulation d'événement discret qui met en oeuvre les principales fonctionnalités requises pour des structures de simulation de haut niveau comme la formation d'une file d'attente et le traitement d'événements, la création de composants système (les services, les machines (Host), le centre de données (Datacenter), le courtier (Broker), les machines virtuelles), la communication entre les composants et la gestion de l'horloge de simulation. Cependant, dans la version actuelle, la couche SimJava a été supprimée afin de permettre à certaines opérations avancées qui ne sont pas pris en charge par celle-ci.

3.3 Résultats expérimentaux

Afin d'évaluer DRAPP, nous avons utilisé le simulateur CloudSim, nous l'avons étendu afin de gérer la réplication des données et la mesure des coûts des ressources. Dans les expériences suivantes, nous avons simulé 4 régions différentes, chaque région contient 2 sous-régions, Une sous-région a 2 ou 3 centres de données (DC), le nombre total de centres de données créés dans cette simulation est 20 DCs. Chaque DC contient 10 hôtes ; Le nombre maximum de machines virtuelles créées dans cette série de simulations est 100. Dans les expériences suivantes, nous mesurons l'impact du budget sur le nombre de répliques et la disponibilité des données. Nous traitons

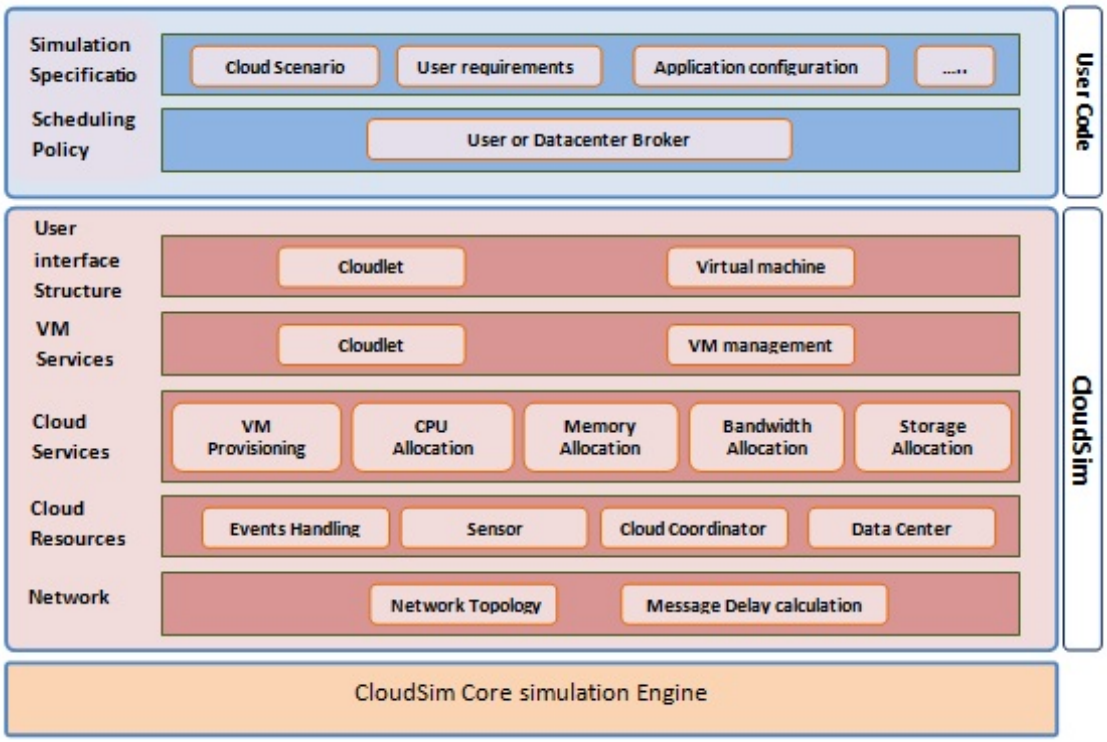


FIG. 3.1 – Architecture du CloudSim [45]

également l'impact des requêtes parallèles sur le nombre de répliques. Ensuite, nous évaluons l'impact des requêtes parallèles sur le temps de réponse et la violation du SLA. Pour ces expériences, nous comparons le résultat de DRAPP à ceux de la stratégie CDRM [18]. Enfin, nous mesurons l'impact du nombre de centres de données sur l'équilibrage de charge et les performances du système.

3.3.1 Impact du budget

A – Effets sur le nombre de répliques

Dans les premières expériences, nous mesurons l'impact des requêtes parallèles sur le facteur de répliques (Figure 3.2). L'impact du budget est également évalué. Nous mesurons le nombre de répliques en variant le nombre de requêtes parallèles. Pour chaque expérience, nous traitons deux budgets : 1 $ et 1,5 $. Nous varions le nombre de requêtes, c'est-à-dire le nombre de Cloudlets, de 10 à 100 requêtes

parallèles qui nécessitent des données différentes. Nous avons remarqué qu'avec les deux budgets le nombre de répliques augmente avec le nombre croissant de Cloudlets. Cela indique que DRAPP ajoute dynamiquement plus de répliques si le nombre de répliques actuel ne satisfait pas les exigences de disponibilité. Le nombre de répliques s'adapte au nombre de requêtes. Il s'adapte également à la valeur du budget. Avec un budget de 1,5 $ par Cloudlet, le nombre de répliques se stabilise plus rapidement. Après avoir atteint une certaine valeur, le nombre de répliques est maintenu à un niveau constant au cours des expériences. Cela signifie que le nombre de répliques actuel est suffisant pour satisfaire les exigences SLO.

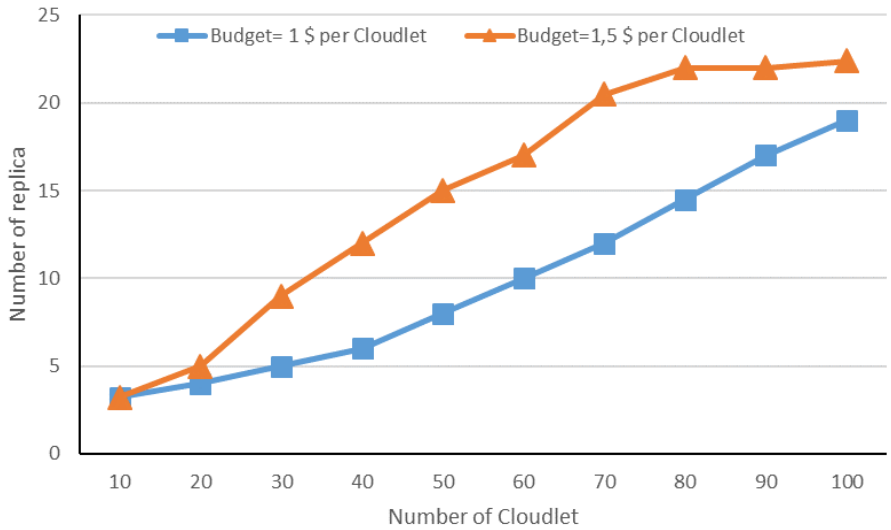


FIG. 3.2 – Impact de la valeur des budgets sur le nombre de répliques

B – Effets sur la disponibilité

Dans cette expérience, nous essayons de voir l'effet du budget sur la disponibilité des répliques (Figure 3.3). Nous avons fixé le nombre de Cloudlets à 60 et nous avons varié le budget total de 10 $ à 80 $. Nous avons calculé la disponibilité des répliques pour chaque valeur budgétaire. Nous notons que la disponibilité augmente considérablement avec l'augmentation du budget et cela est dû à l'augmentation du nombre de répliques. Par exemple, lorsque le budget total est de 40, la disponibilité est de 95%.

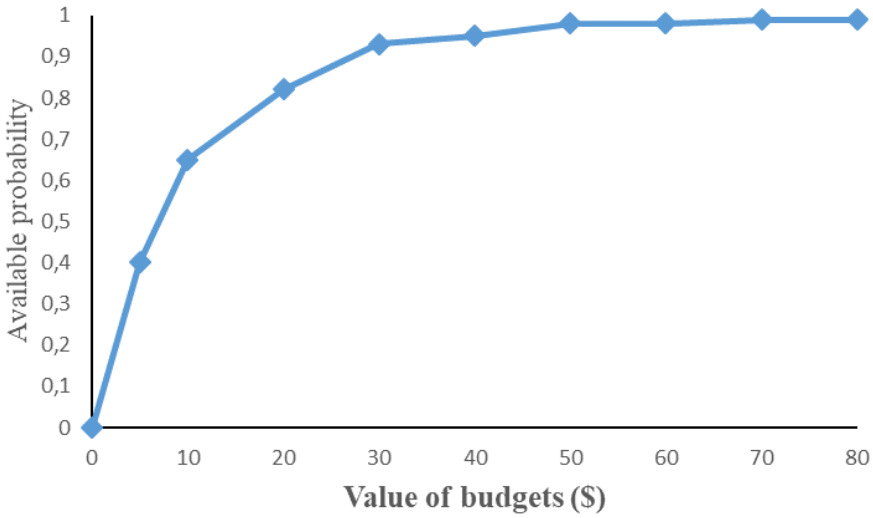


FIG. 3.3 – Impact de la valeur des budgets sur la disponibilité

C – Effets sur le temps de réponse

Le but de la simulation suivante est d'étudier l'influence de la valeur du budget sur le temps de réponse moyen ; Nous mesurons le temps de réponse moyen de toutes

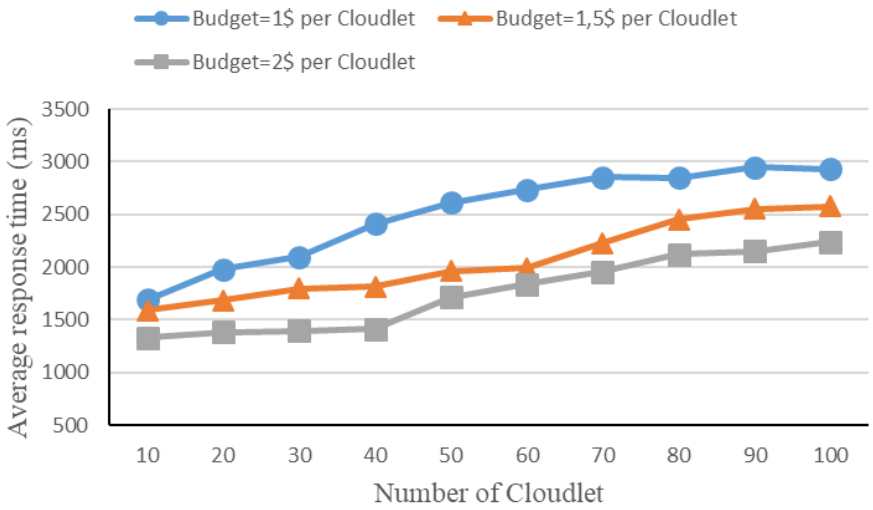


FIG. 3.4 – Impact de la valeur des budgets sur le temps de réponse

les requêtes traitées pendant la simulation avec différents budgets (1 $, 1,5 $ et 2 $). Nous avons varié le nombre de Cloudlet de 10 à 100 requêtes. Comme le montre la Figure 3.4, les meilleurs temps de réponse sont obtenus avec le plus gros budget. C'est tout à fait normal puisque le budget influe sur le nombre de répliques puis sur le temps de réponse. Si la valeur du budget est faible, nous avons moins de répliques. Ensuite, le temps d'attente augmente. Par conséquent, le temps de réponse moyen augmente.

3.3.2 Impact du nombre de centres de données

Dans les expériences suivantes, nous mesurons l'impact du nombre de DCs sur l'équilibrage de charge du système. Nous effectuons la simulation avec 5 régions où chaque région contient deux sous-régions. Pour la première expérience, nous avons créé 2 DCs dans chaque sous-région, soit un total de 20 DCs. Dans d'autres expériences, nous traitons 3 DCs par sous-régions, soit un total de 30 DCs. Le nombre total de VMs créées dans cette série d'expériences est de 60.

A – Effet sur l'équilibrage de la charge du système

L'emplacement des répliques et la planification des requêtes ont un impact significatif sur l'équilibrage de charge. Dans cette expérience, nous comparons le pour-

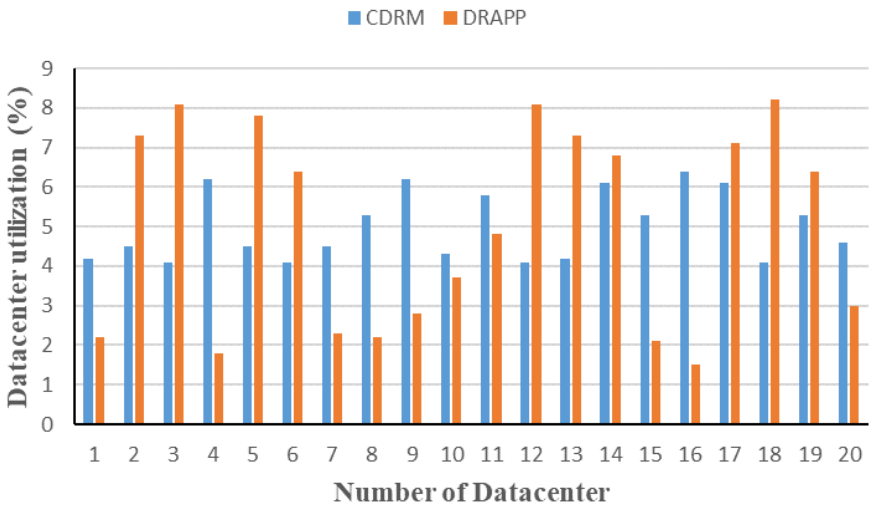


FIG. 3.5 – Impact du nombre de centres de données sur l'équilibrage de charge

centage de charge sur différents centres de données. Nous utilisons 20 DCs. Ensuite, nous calculons le pourcentage d'utilisation de chaque DC. Le résultat du test est représenté sur la Figure 3.5. Nous pouvons voir qu'avec CDRM, nous avons un bon équilibre d'utilisation entre les différents DC. Par exemple, l'écart entre le centre de données le plus chargé et le moins chargé est seulement 2,3%. Cet écart reflète un bon équilibrage de charge. En effet, le CDRM répartit uniformément la charge de travail déséquilibrée sur tous les DCs en utilisant la migration de données.

B – Effet sur le temps de réponse

Dans cette expérience, notre objectif est de voir comment l'augmentation du nombre de DCs peut affecter le temps de réponse moyen des requêtes (Figure 3.6). Nous avons varié le nombre de DCs par pas de 5 et nous avons fixé le nombre de requêtes à 60. Dans un premier temps, nous notons un léger avantage de la stratégie CDRM par rapport au DRAPP. Mais quand le nombre de DCs passe à 10, on remarque que DRAPP donne de meilleures performances. Cet avantage devient de plus en plus clair avec l'augmentation du nombre de DCs. Nous pouvons expliquer ce résultat par la façon de distribuer les requêtes. Dans DRAPP, nous classons les requêtes en fonction de leur région. Ensuite, nous assignons ces requêtes aux DCs dans la même région. Cela minimise le temps de réponse.

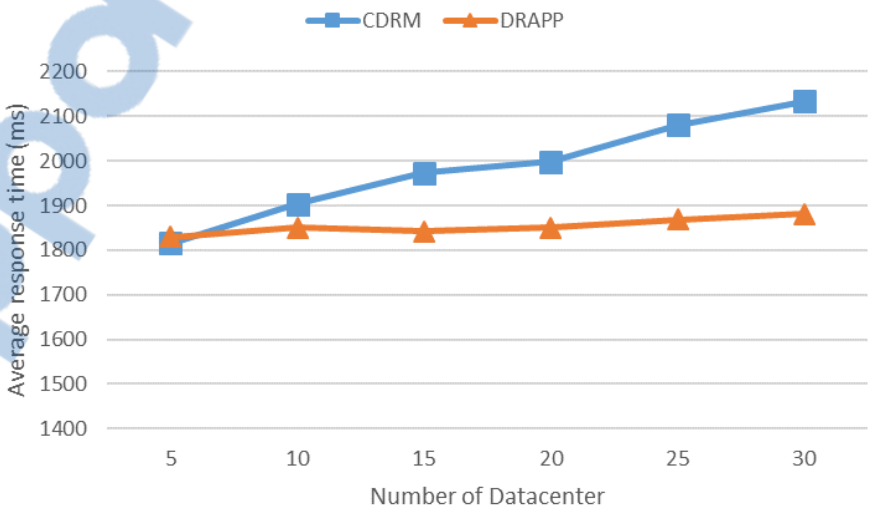


FIG. 3.6 – Impact du nombre de centres de données sur le temps de réponse

C – Effets sur la consommation de bande passante

Dans cette expérience, nous mesurons la consommation de bande passante pour les stratégies DRAPP et CDRM. Le nombre de DCs varie de 5 à 30 et le nombre de requêtes, c'est-à-dire le nombre de Cloudlet, a été fixé à 60. La Figure 3.7 illustre la consommation de bande passante avec les deux stratégies comparées. Ces résultats montrent que la consommation de bande passante avec la stratégie CDRM est plus élevée. Le CDRM utilise la migration pour équilibrer la charge entre les différents DCs, ce qui influence la consommation de la bande passante. La consommation de la bande passante avec le DRAPP est moins importante car les répliques sont plus proches. Ensuite, moins de transfert de données est nécessaire, ce qui a pour effet de réduire la consommation de bande passante.

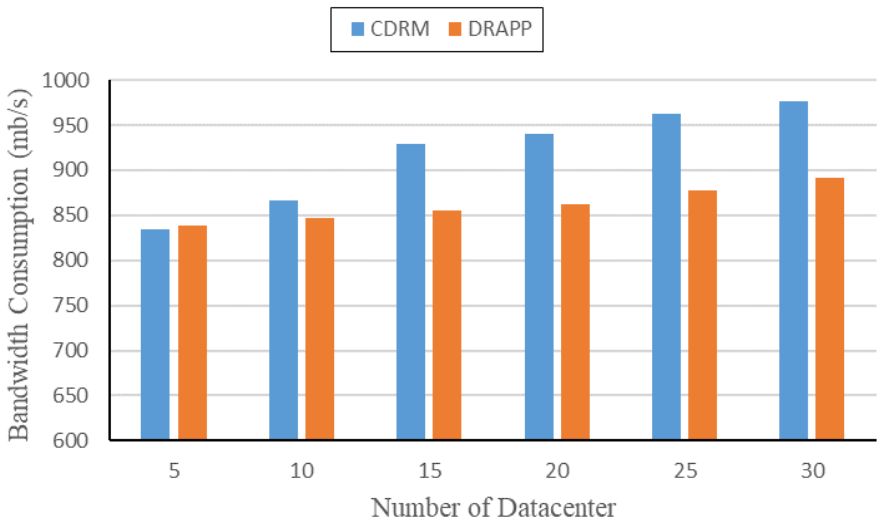


FIG. 3.7 – Impact du nombre de centres de données sur la consommation de bande passante

3.3.3 Impact du nombre de requêtes

Dans ces expériences, nous évaluons l'impact du nombre de requêtes sur les performances et la violation SLA. Nous gardons les mêmes paramètres de simulation des expériences précédentes.

A – Effets sur le temps de réponse

Pour étudier l'impact du nombre de Cloudlets sur le temps de réponse moyen, nous avons varié le nombre de requêtes de 10 à 100 et nous avons mesuré le temps de traitement des requêtes comme le montre la Figure 3.8. Le nombre de centres de données a été fixé à 20. Nous remarquons une diminution significative du temps de réponse pour les requêtes expérimentées avec le DRAPP par rapport à la stratégie CDRM. DRAPP s'adapte au nombre de requêtes. Une nouvelle réplique est créée uniquement si le gain du fournisseur est réel. Une réplique est placée dans l'hôte qui reçoit le plus de requêtes en opposition de CDRM qui utilise la migration qui constitue un surcoût affectant le temps de réponse de la requête. Par conséquent, le temps de transfert et le temps de réponse sont réduits.

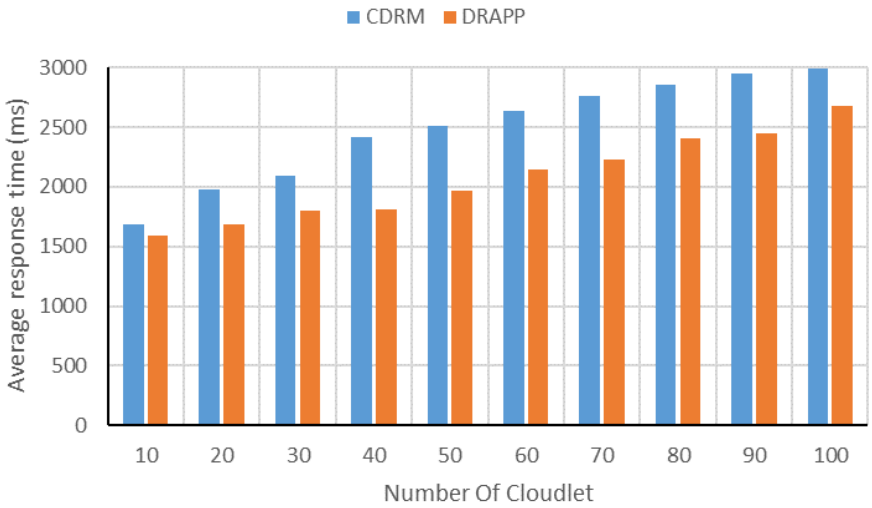


FIG. 3.8 – Impact du nombre de cloudlets sur le temps de réponse

B – Effets sur la violation du SLA

Une caractéristique importante qui différencie le Cloud aux modèles commerciaux traditionnels est le mécanisme de pénalité. En cas de violation d'un contrat SLA, le fournisseur est tenu de verser au locataire une somme d'argent convenue [115]. L'expérience suivante vise à vérifier le nombre de violations de SLA. Nous avons

basé sur les mêmes paramètres des expériences précédentes. Pour calculer le nombre de violations SLA, nous vérifions le temps de réponse de chaque requête. Ensuite, nous le comparons au seuil de temps de réponse défini dans le SLA. Si le temps de réponse est supérieur à ce seuil, il existe une violation SLA. Selon les résultats présentés dans la Figure 3.9, nous avons remarqué que le nombre de violations de SLA est très faible parce que DRAPP est dynamique. Si le nombre de répliques à un moment donné provoque une violation SLA, alors de nouvelles répliques sont créées. En outre, lorsque le coût de création de nouvelles répliques dépasse le budget, la création d'une réplique est impossible. Cela justifie l'existence de certains cas de violation des SLA.

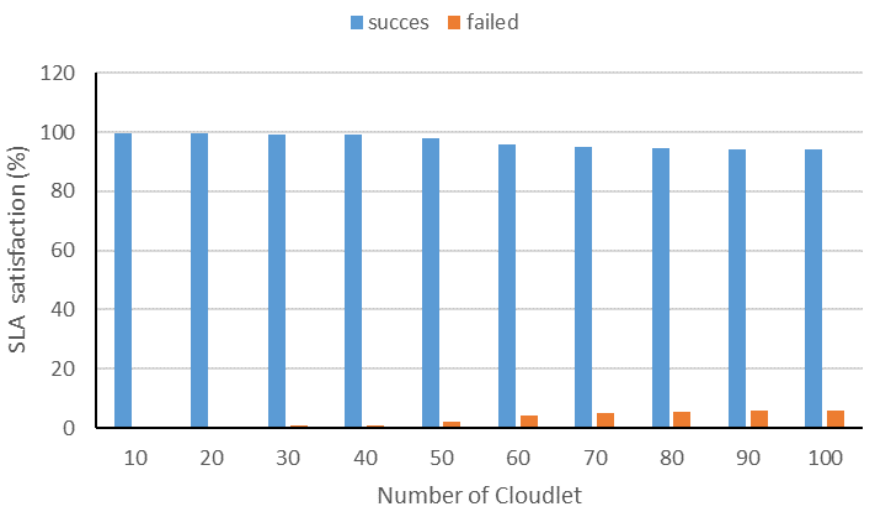


FIG. 3.9 – Impact du nombre de cloudlets sur la violation du SLA

3.4 Conclusion

Dans ce chapitre nous avons largement étendu un simulateur de Cloud existant, CloudSim [47], que nous avons déjà partiellement étendu dans [19] afin d'évaluer notre stratégie DRAP. Nous avons réalisé plusieurs simulations en jouant sur différents paramètres comme : le nombre de requêtes, le nombre de centres de données et le budget. Nous avons étudié l'impact de ces paramètres sur plusieurs métriques

comme le temps de réponse, la bande passante, la disponibilité de la donnée, etc. Les résultats de simulations montrent que la stratégie proposée peut améliorer significativement la disponibilité et la performance simultanément tout en prenant en compte la rentabilité du fournisseur de Cloud.

Deuxième partie

Gestion de la cohérence de données

CHAPITRE 4

Etat de l'art sur les stratégies de cohérence de données

4.1 Introduction

La réplication de données peut être utilisée pour améliorer la disponibilité et pour éviter la dégradation des performances due à la surcharge des nœuds hébergeant les données demandées. Mais elle introduit un surcoût supplémentaire pour maintenir la cohérence des répliques. En conséquence, ce surcoût peut affecter la performance et la disponibilité du système. Dans la littérature, on distingue deux grandes familles de cohérence : cohérence forte et cohérence faible. La cohérence forte évite toute divergence entre les différentes répliques de même objet, tandis que la cohérence faible est utilisée pour éviter le surcoût de la cohérence forte, elle permet une certaine différence entre les répliques. Certains systèmes combinent la cohérence faible et la cohérence forte pour maintenir une latence faible. Dans ce chapitre nous allons étudier quelques travaux qui gèrent la cohérence de données dans les systèmes à grande échelle. Dans un premier temps nous citons quelques stratégies de cohérence de données dans l'environnement des grilles, puis nous classons les stratégies les plus référencées dans les Clouds.

4.2 Stratégies de cohérence dans les grilles

La cohérence des données est le principal problème de la gestion des données modifiables. C'est un domaine de recherche actif dans les environnements des grilles.

Dans [116], Belalem et al. décrivent un modèle à double couches adapté aux applications à grande échelle et qui représente le support de l'approche hybride de la gestion de la cohérence des répliques basée sur des approches pessimistes et optimistes. Cette approche hybride présente un mécanisme adapté basé sur les différentes formes de négociation entre les agents de cohérence virtuelle afin de réduire le nombre de conflits entre les répliques dans les grilles de données.

Un autre modèle à deux niveaux a été proposé aussi par Domenici et al. [117] ; ils ont proposé un service de cohérence de répliques (RCS), qui a été développé par le projet européen de grille de données. La responsabilité de RCS est de maintenir la cohérence entre les répliques existantes. Pour garantir la cohérence locale, le service de cohérence locale est exécuté dans chaque unité de stockage. Un mécanisme de verrouillage de fichier est également proposé dans RCS pour la sérialisation de l'accès au fichier. Cependant, les auteurs n'ont fourni aucun détail sur le processus de détection des situations de divergence ou de conflit entre les répliques. En plus, les deux services de cohérence locale et globale ont été brièvement expliqués, les auteurs n'ont pas détaillé comment ces services peuvent être utilisés pour garantir une cohérence entre les répliques.

Lorsque les mises à jour de données sont autorisées, la gestion des activités d'accès aux données est très importante afin de préserver la cohérence et la fiabilité des données. Dans le travail [118], Abawajy et al. formulent le problème de réplication de données et conçoivent un algorithme de réplication de données distribué avec une garantie de cohérence pour la grille de données. L'approche consiste à organiser systématiquement les sites de la grille de données en des régions distinctes. Les principales contributions de ce travail sont : i) la proposition d'une nouvelle politique de placement des répliques. ii) la proposition d'un nouveau protocole de réplication de données basé sur le quorum ; Le quorum sert d'outils de base pour fournir un moyen uniforme et fiable pour assurer la cohérence entre les répliques du système. iii) l'étude des divers compromis en termes de coût, de disponibilité et de complexité algorith-

mique du schéma de réplication proposé. L'inconvénient majeur de cette stratégie proposée est l'utilisation du mécanisme de quorum pour garantir la cohérence, ce qui réduit la performance du system et le rend moins évolutif.

Dans le travail [26], Belalem et al. ont comparé la cohérence pessimiste et la cohérence optimiste, ils ont proposé une solution qui combine ces deux approches pour garantir la cohérence entre les répliques dans les grilles. La cohérence optimiste est utilisé pour assurer la cohérence des répliques dans chaque site. Alors que la cohérence globale est garantie par l'utilisation d'un algorithme inspiré de l'approche pessimiste. L'utilisation d'une cohérence optimiste sur les sites locaux peut entraîner des conflits entre les répliques des sites ; Cependant, la stratégie proposée gère la résolution des conflits sur les sites locaux seulement et ne prennent pas en charge la résolution des conflits des répliques entre les sites.

4.3 Stratégies de cohérence dans les Clouds

4.3.1 Stratégies de cohérence faible

Plusieurs Stratégies ont été proposées dans différents contextes pour spécifier, vérifier, mesurer et prédire la cohérence faible. Dans cette partie, nous proposons quelques travaux qui utilisent la cohérence faible pour assurer la cohérence entre les répliques dans l'environnement du Cloud.

Les systèmes de base de données NoSQL sont conçus pour atteindre un haut débit et une haute disponibilité en abandonnant certaines fonctionnalités offertes par les systèmes de bases de données traditionnels telles que les jointures et les transactions ACID. Les bases de données NoSQL peuvent offrir des propriétés de cohérence plus faibles, par exemple une cohérence éventuelle. Un client d'une telle base de données peut lire des valeurs anciennes. Cette caractéristique de conception est expliquée par le théorème CAP, qui indique que pour une base de données répartie et répliquée, il est possible de fournir au plus deux des trois propriétés suivantes : cohérence des répliques, disponibilité des données et tolérance de partition. De nombreux systèmes de base de données NoSQL visent la disponibilité et la tolérance de partition comme objectif principal et ils assouplissent donc les contraintes de cohérence des données. Dans [119], Wada et al. ont fait une étude détaillée sur plusieurs plateformes de sto-

ckage, qui montrent à quelle fréquence et dans quelles circonstances, les différentes situations d'incohérence sont observées et quel est l'impact de choisir de fonctionner avec des mécanismes de cohérence faibles sur les propriétés de performance. Dans un autre travail [120], Golab et al. proposent une nouvelle métrique de cohérence appelée Γ(Gamma) qui capture la cohérence observée par le client. Cette métrique fournit des réponses quantitatives aux questions concernant les anomalies de cohérence observées, telles que la fréquence à laquelle elles se produisent et leur gravité lorsqu'elles se produisent. Les auteurs soutiennent que Γ est plus utile et plus précis que les autres mesures existantes. Ils ont montré expérimentalement que leur technique centrée sur le client présente des sensibilités substantielles à une variété de paramètres de configuration et de charge de travail, plus important encore, la taille des clés (the keys-pace size), les niveaux de cohérence en lecture/écriture côté client, la distribution des clés et le facteur de réplication. En comparaison, les mesures d'une métrique de cohérence centrée sur le système montraient une sensibilité faible ou nulle et indiquaient que des anomalies de cohérence substantielles pouvaient survenir même dans les cas où aucune anomalie de ce type n'était observée avec la métrique Γ.

Le travail [121] explore le problème de la cohérence causale pour les systèmes partiellement répliqués où les auteurs présument qu'il est avantageux d'utiliser une réplication partielle des données. Ils ont proposé deux algorithmes pour obtenir une cohérence causale dans les systèmes Clouds sous l'hypothèse d'utiliser des données partiellement répliquées. L'algorithme Full-Track peut mettre à jour les répliques locales dès que possible tout en respectant la cohérence causale, ce qui permet de réduire la fausse causalité dans le système. L'algorithme Opt-Track est une amélioration de l'algorithme Full-Track qui a été rendu plus optimal en réduisant la taille des journaux locaux maintenus. En plus, il minimise la quantité d'informations de contrôle ajoutées sur les messages de mise à jour. Ils proposent également l'algorithme Opt-Track-CRP dans le cas de la réplication complète.

L'implémentation du Cloud de données est opaque pour tous les utilisateurs en raison de la technique de virtualisation. Ainsi, il est difficile pour les utilisateurs de vérifier si chaque réplique dans le Cloud de données est la dernière ou non. Inspirés par la solution de [13], Liu et al. [122] permettent aux utilisateurs du Cloud d'audit de

vérifier la cohérence du Cloud en analysant une trace d'opérations interactives. Une structure d'audit à deux niveaux est adoptée : chaque utilisateur peut effectuer un audit local avec une trace locale des opérations ; un auditeur est élu périodiquement pour effectuer un audit global avec une trace globale des opérations.

Le travail [123] s'intéresse au problème de la maintenance de cohérence entres les répliques des bases de données dans le Cloud. Les auteurs proposent une approche de cohérence arborescente qui réduit l'interdépendance entre les serveurs de répliques en introduisant des états partiellement cohérents et parfaitement cohérents des bases de données. L'arborescence est formée de telle sorte que le chemin d'accès le plus fiable soit assuré depuis le serveur principal vers tous les serveurs de réplication. En conséquence, la probabilité d'échec d'une transaction est considérablement réduite, ce qui permet d'augmenter les performances et conservé le débit, même dans un réseau non fiable.

Dans le même contexte, Balegas et al. [124] ont présenté un middleware qui permet de garantir une base de donnée cohérente dans le Cloud ; l'approche proposée est basée sur une extension de la cohérence éventuelle. Elle ne contient aucune autorité centrale et elle est totalement asynchrone. les auteurs ont combiné la cohérence éventuelle qui a comme avantage une faible latence et une haute disponibilité, avec ceux d'une cohérence forte, en appliquant des invariants numériques globaux. En plus, le middleware proposé combine la mise en cache avec le traitement par lots, ce qui améliore le débit d'écriture sans réduire les propriétés de tolérance aux pannes du système.

4.3.2 Stratégie de cohérence adaptative

Pour offrir une haute disponibilité et des performances accrues, différents schémas de réplication sont utilisés pour maintenir la cohérence entre les répliques. Avec les modèles de cohérence classiques, les performances sont nécessairement dégradées. De plus, l'exigence de cohérence de certaines applications change lors de l'exécution, donc une stratégie de cohérence fixe n'est pas suffisante. Une solution consiste à combiner plusieurs modèles de cohérence qui permet l'utilisation de niveau de cohérence approprié en fonction des besoins de l'application / de l'utilisateur, cette pratique est appelée cohérence adaptative. Pour aborder le compromis entre disponibilité et

cohérence, Esteves et al. [125] ont proposé l'utilisation d'un nouveau modèle de cohérence avec un Framework nommé VFC^3(Versatile Framework for Consistency in Cloud Computing)et un support de bibliothèque de programmation qui permet la définition et l'application dynamique de plusieurs degrés de cohérence sur différents groupes de données. Dans ce modèle les niveaux de cohérence peuvent être ajustés automatiquement en fonction des informations statistiques. Les résultats de simulations montrent que VFC^3 est efficace pour améliorer la QoS, il permet de réduire la latence, la bande passante ; ceci, tout en maintenant les exigences relatives à la qualité des données fournies aux utilisateurs.

Dans un autre travail, Wang et al. [126] ont proposé une stratégie de cohérence appropriée est adoptée en fonction de l'état de fonctionnement dynamique du système. La stratégie proposée fait attention à établir un bon équilibre entre la cohérence, la disponibilité et la performance. Cette stratégie permet au système de changer automatiquement le modèle de cohérence en fonction de la fréquence de mise à jour et de la fréquence de lecture d'un fichier. Cependant, l'ajustement des paramètres n'est pas une tâche simple. Les auteurs proposent d'améliorer ce travail en prenant en compte plus de paramètres pour sélectionner la stratégie de cohérence, par exemple en fonction de l'état de stockage et les exigences spécifiées par l'application.

Dans le même contexte, Chihoub et al. ont proposé une nouvelle approche [127], appelée "Harmony", qui règle automatiquement le niveau de cohérence au moment de l'exécution en fonction des exigences de l'application. Harmony surveille le système de stockage et les accès aux données afin d'estimer le taux de lectures périmées dans le système. En conséquence, il augmente/réduit le niveau de cohérence pour préserver un taux de vétusté toléré par l'application. L'approche proposée ne donne aucune garantie sur le moment où toutes les répliques du système de stockage convergeraient vers un état cohérent. Cela pourrait être un réel problème pour de nombreuses applications car il n'y a aucune certitude à propos des données lues. En outre, le système proposé pourrait être amélioré en ajoutant différents niveaux de garanties en tenant compte de la performance et de la topologie du réseau, en plus de la localisation des données. Ces améliorations ont été proposées dans un autre travail [128].

Un travail similaire à [127] a été proposé dans [129], Zhou et al. ont présenté

une stratégie de cohérence de répliques auto-adaptative qui permet au système de sélectionner automatiquement la stratégie de cohérence adéquate en fonction de la chaleur du fichier (le fichier qui a subit un accès fréquent). En se basant sur l'impact de la séquence de temps d'accès au fichier. La chaleur du fichier calculée permettant de prédire avec précision le comportement d'accès au futur fichier. Ils introduisent un nouvel algorithme MRFU (Most Recently and Frequently Used) pour calculer la chaleur d'un fichier. L'idée est d'attribué la plus grande valeur de chaleur au fichier utilisé le plus récemment et fréquemment dans une période de temps. Les auteurs utilisent plusieurs répliques principales et plusieurs sous-répliques, ils n'ont pas traité le cas de la modification de plusieurs sous-répliques en même temps, et l'approche proposée ne prend pas en charge la cohérence globale. Ils n'ont pas décrit comment résoudre les conflits dans le cas de l'utilisation d'une stratégie de cohérence éventuelle.

Dans [130], Kim et al. ont présenté un modèle de cohérence adaptative nommé ACM (Adaptive Consistency Model). L'objectif de l'ACM est de gérer de manière dynamique et élastique la cohérence au moment de l'exécution, afin de fournir des compromis adéquats entre la cohérence, la performance et la disponibilité. En conséquence, ACM considère non seulement les exigences de l'application mais également l'état du système de stockage. De plus, il ne s'appuie pas sur un modèle standard basé uniquement sur le modèle d'accès pour définir l'exigence de cohérence d'une application. Les auteurs n'ont pas réussi à montrer comment choisir le niveau de cohérence et ils n'ont présenté aucune formule pour calculer le seuil. En outre, l'utilisation de multiples niveaux de cohérence implique la présence de conflits entre les répliques ; les auteurs n'ont pas mentionné comment résoudre ce problème.

Une autre approche de cohérence des données à plusieurs niveaux pour gérer les données des services SaaS a été proposée dans [131]. Les développeurs de services définissent dynamiquement leurs exigences de cohérence selon leurs besoins sous la forme d'un plan de cohérence des données appelé DCP (Data Consistency Plan), puis soumettent ce plan à la plate-forme DCaaS (Data Consistency as a Service), qui s'assurera que ces exigences de cohérence sont satisfaites lors des opérations d'accès aux données. L'approche proposée est basée sur les quotas pour assurer l'exactitude des données globales dans le Cloud. Cette contribution garantit la cohérence globale des données en répartissant la capacité des objets de données de forte cohérence entre

les centres de données, puis elle adopte une approche de réplication paresseuse pour synchroniser les entrepôts de données. L'inconvénient de l'approche est que le niveau de cohérence est sélectionné manuellement ; Les développeurs de services définiront les exigences de cohérence pour chaque objet de données.

L'incapacité à gérer le compromis entre la cohérence et l'évolutivité peut conduire les systèmes Cloud à violer certaines exigences de cohérence. Dans ce contexte, Chen et al. [132] ont présenté une approche et une architecture orientée services qui peuvent être utilisées pour s'adapter à différents modèles de cohérence au moment de l'exécution et en fonction des exigences prédéfinies. Ils ont introduit la notion de « région de cohérence», une région de cohérence est un domaine logique qui utilise des modèles de cohérence et qui gère de manière autonome les exigences de cohérence pour un groupe de service(s). Ils ont proposé deux protocoles, Multi-fixed Sequencer Protocol (MSP) et Region-based Election Protocol (REP). MSP vise à garantir la satisfaction des modèles de cohérence dans une région et REP est responsable de l'équilibrage de la charge de travail entre les séquenceurs. La solution proposée favorise une cohérence flexible et améliore l'évolutivité des applications basées sur le Cloud.

La nécessité des stratégies qui gère la cohérence d'une façon adaptative a été identifiée dans [132], [125], [131], [126] où les auteurs définissent différents niveaux de cohérence qui peuvent être ajustés automatiquement. Cependant, ces approches ne traitent pas le problème des coûts liés à la cohérence des données dans le Cloud. En plus de la performance, les coûts sont également un facteur important dans le Cloud. En général, plus le niveau de cohérence est élevé, plus les coûts de mise en œuvre sont élevés et plus le degré d'évolutivité est faible. Une cohérence faible est moins coûteuse mais peut entraîner des coûts d'incohérence et elle conduit à des pertes d'opératoires élevées. Dans ce contexte, Fetai et al. [133] proposent un contrôle concurrentiel basé sur le coût nommé C3 (Cost-based Concurrency Control), qui est une approche du contrôle de la concurrence dans les environnements Cloud DaaS (Data-as-a-Service). Les principales caractéristiques de C3 sont son comportement adaptatif. Avec C3, le coût global de l'application sera minimisé. L'approche C3 est basée sur un modèle de coût générique pour la cohérence des données et une API de programmation intuitive et puissante qui permet aux utilisateurs de spécifier les paramètres de coût au niveau de la transaction. Sur la base de ces paramètres, C3

ajustera le niveau de cohérence de la transaction afin de minimiser les coûts globaux. Les auteurs analysent les anomalies possibles en cas de mélanges de cohérence et fournissent un mécanisme pour éviter ces anomalies.

Un travail similaire a été présenté dans [128], Chihoub et al. ont proposé des méthodes auto-adaptatives qui ajustent les niveaux de cohérence au moment de l'exécution afin d'obtenir de meilleures performances, une meilleure disponibilité et réduire le coût monétaire sans violer les exigences de cohérence de l'application. En outre, ils introduisent une méthode de modélisation du comportement qui analyse automatiquement l'application et apprend ses exigences de cohérence.

Dans le même contexte, Chihoub et al. [134] étudient le coût monétaire de la cohérence dans le Cloud. Leur analyse détaillée et leur étude ont révélé une variation notable des coûts monétaires lorsque différents niveaux de cohérence sont utilisés. Dans un premier temps pour comprendre les impacts de différentes cohérences sur le coût monétaire et les nouvelles lectures dans le Cloud, la métrique d'efficacité du coût de cohérence a été définie. Sur la base de cette métrique, ils introduisent une approche simple mais efficace, nommée Bismar, qui ajuste de façon adaptative le niveau de cohérence au moment de l'exécution afin de réduire le coût monétaire tout en maintenant une faible fraction de lectures périmées. Bismar s'appuie sur un modèle probabiliste de cohérence qui estime les lectures périmées et les coûts relatifs de l'application en fonction du taux de lecture/écriture actuel et de la latence du réseau. Bismar a été implémentée avec des évaluations intensives sur le système de stockage Cloud Cassandra sur Grid'5000. Les résultats montrent que Bismar peut conduire à un coût efficace sans dépasser le nombre toléré de lectures périmées sur les applications.

Le défi principal abordé dans [135] est de savoir comment définir et implémenter des SLA basés sur la cohérence qui offrent un ensemble étendu de niveaux de service lors de l'accès à un entrepôt de clés/valeurs évolutif. Tout d'abord, les auteurs présentent la conception d'un système de stockage Cloud avec une gamme de choix de cohérence qui se situent entre la cohérence forte et éventuelle nommé Pileus. Ce dernier permet à différentes applications (ou différents utilisateurs d'une application) d'obtenir différentes garanties de cohérence même en partageant les mêmes

données. Deuxièmement, ils introduisent la notion de SLA qui intègre la cohérence ainsi que la latence. Grâce aux choix de cohérence qu'ils proposent, le SLA d'une application peut indiquer un large éventail de compromis acceptables en matière de cohérence/latence. Les auteurs proposent d'exprimer ces SLA basés sur la cohérence comme une série de choix alternatifs avec une valeur décroissante pour l'application. Ils présentent et évaluent des techniques qui tentent de maximiser la valeur délivrée lors de la lecture des données.

Dans un système de base de données transactionnelle qui est déployé dans un système hautement distribué et élastique tel que le Cloud, les bases de données sont généralement répliquées sur plusieurs sites. l'utilisation du même modèle de cohérence faible ou éventuel pour gérer la cohérence peut provoquer des décisions dangereuses en lisant des données périmées. Dans ce contexte, Iskander et al. [136] ont défini les notions de transactions fiables et sécurisées, ils ont introduit aussi différents niveaux de cohérence et ils ont présenté différentes preuves d'approches d'autorisation pour réaliser des transactions de confiance. Chaque approche offre différentes garanties de confiance au cours de l'exécution des transactions. En outre, les auteurs ont proposé une solution qui implique une adaptation du protocole de validation en deux phases nommé 2PVC (2-Phase Validation Commit protocol) pour appliquer les transactions de confiance.

La réduction de la latence pour les protocoles de validation transactionnels est l'objectif d'un autre travail [137]. Kraska et al. décrivent MDCC (Multi-Data Center Consistency), un protocole de validation optimiste pour les transactions dont le coût est similaire à celui des protocoles éventuellement cohérents. MDCC n'a besoin que d'un seul aller-retour de message étendu pour valider une transaction dans le cas commun. Ce qui signifie qu'il peut lire ou mettre à jour à partir de n'importe quel centre de données. Le protocole de validation MDCC peut être combiné avec différents niveaux d'isolation qui garantissent des propriétés variables pour la récence et la cohérence mutuelle des opérations de lecture.

4.4 Conclusion

Dans ce chapitre nous avons cité quelques travaux réalisés dans le cadre de la gestion de la cohérence des données dans les systèmes à grande échelle. Dans la litté-

rature nous trouvons plusieurs travaux qui se sont intéressé à l'utilisation de plusieurs degrés de cohérence pour faire un compromis entre la cohérence et la disponibilité. Peu de travaux proposent un ajustement automatique entre les niveaux de cohérence. En outre, l'utilisation de multiples niveaux de cohérence implique la présence de conflits entre les répliques d'un même objet ; les chercheurs n'ont pas mentionné comment résoudre ce problème. Ils ne se contentent que de détecter les conflits. C'est la raison pour laquelle, nous proposons une stratégie hybride et adaptative qui utilise les vecteurs de version pour gérer la cohérence de données. Nous étendons notre stratégie par l'intégration d'un mécanisme de convergence et de résolution de conflits entre les répliques.

CHAPITRE 5

Stratégie proposée pour gérer la cohérence de données

5.1 Introduction

La réplication de données a été largement adoptée dans les systèmes à grande échelle pour améliorer la fiabilité et la performance. Mais avec l'utilisation de la réplication de données apparaît souvent le problème de la cohérence de données. Bien que le maintien d'une forte cohérence entre les répliques puisse garantir le bon fonctionnement des applications, cependant, cela affectera la performance et l'évolutivité du système. Parce que les données doivent être verrouillées pendant la période de mise à jour pour s'assurer qu'aucun autre processus ne modifie les mêmes données. De nombreuses applications favorisent les performances en utilisant un modèle de cohérence faible. Mais il y a une forte possibilité de lire des données qui ne sont pas à jour, ce qui conduit à des résultats incorrects. Afin de contribuer à la résolution du problème de gestion de cohérence des données dans le Cloud Computing, nous proposons dans ce chapitre une stratégie appelée AC2R (Adaptive Consistency strategy with Conflict Resolution). La stratégie AC2R est basée sur un modèle hybride qui utilise les vecteurs de version entre les répliques. Puis, nous améliorons AC2R par l'intégration d'un mécanisme de convergence et de résolution de conflits entre les répliques.

5.2 Topologie de Cloud

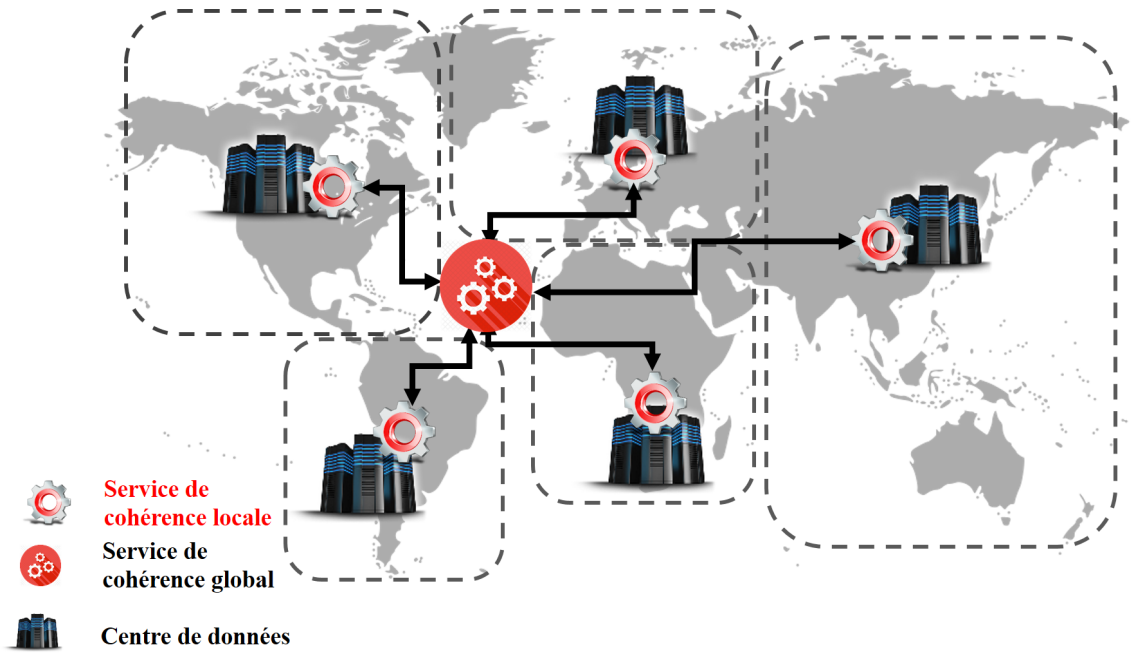


FIG. 5.1 – Modèle hiérarchique pour la gestion de la cohérence des répliques.

La stratégie AC2R est basée sur la modélisation d'un Cloud sous la forme d'un modèle hiérarchique à deux niveaux (voir Figure 5.1). Chaque centre de données contient un ensemble d'hôtes et chaque hôte est composé essentiellement de deux types de ressources : les éléments de calcul (CE) et les éléments de stockage (SE). Les données répliquées sont stockées sur SE et accessibles à partir de CE via des opérations de lecture ou d'écriture. Chaque réplique associée à des informations supplémentaires est appelée métadonnées. Ce dernier donne une description détaillée de la réplique et peut contenir plusieurs informations supplémentaires sur l'état de cette réplique, par exemple le nombre de versions, timestamp (l'horodatage), l'origine de la mise à jour, le facteur de priorité, etc. Nous définissons un service de cohérence locale (LCS) pour chaque centre de données. Un service de cohérence locale LCS_i est responsable de : i) Gérer la cohérence des répliques dans un centre de données DC_i. ii) Coopérer avec les autres LCS_j pour assurer une cohérence de donnée pour l'ensemble du Cloud.

5.3 Service de cohérence locale

Le service de cohérence local est constitué d'un groupe de trois agents coopérants : un agent de surveillance, un agent de cohérence et un agent de gestion des conflits.

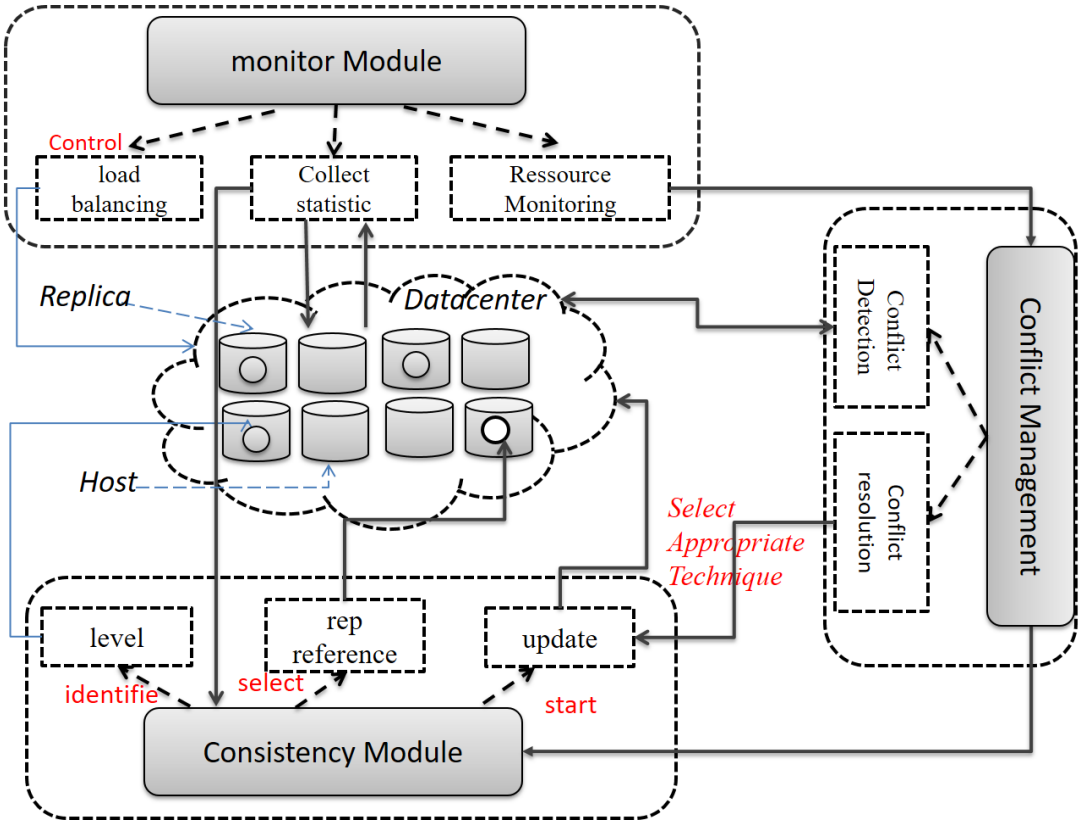


FIG. 5.2 – Architecture du service de cohérence locale

5.3.1 Agent de surveillance

Cet agent collecte les informations pertinentes nécessaires pour gérer la cohérence locale. Il collecte le nombre de lectures et d'écritures dans les éléments de stockage. L'agent de surveillance a été conçu de manière parallèle afin de le rendre plus efficace et de réduire le temps de surveillance. Chaque processus collecte les statistiques à partir d'un hôte et à la fin un processus d'agrégation est appliqué pour calculer les taux de lecture et d'écriture. Ces données sont en outre communiquées à l'agent de

cohérence. L'agent de surveillance est également responsable d'observer la charge du centre de données (voir Figure 5.2).

5.3.2 Agent de cohérence

Les exigences de cohérence varient d'une application à une autre, l'agent de cohérence utilise trois niveaux de cohérence pour un centre de données, ces trois niveaux sont déterminés en fonction de la fréquence d'écriture. L'agent de cohérence utilise les informations collectées par l'agent de surveillance afin de sélectionner le niveau de cohérence adéquat à l'application. Deux paramètres sont utilisés : les seuils d'écriture $Tmin$ et $Tmax$, ces paramètres déterminent la marge pour basculer entre les trois niveaux comme indiqué sur la Figure 5.3.

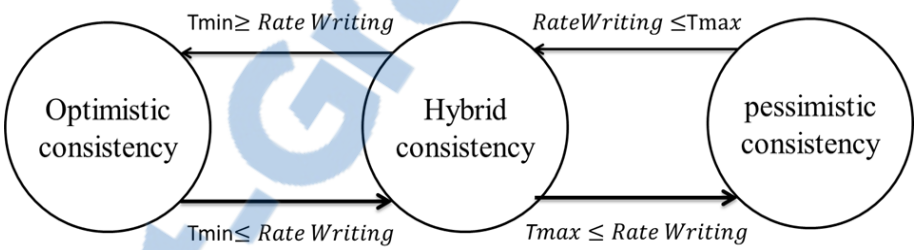


FIG. 5.3 – Niveaux de cohérence du centre de données

Cohérence optimiste

Cette stratégie tolère une certaine divergence entre les répliques. Un utilisateur peut accéder à une réplique qui n'est pas nécessairement à jour. Cette stratégie se focalise sur l'amélioration des performances. Le taux d'écriture dans cette stratégie est inférieur au seuil $Tmin$;

Cohérence hybride

Cette stratégie fournit un compromis adéquat entre la performance et la cohérence. L'agent de cohérence sélectionne ce niveau de cohérence lorsque le taux d'écriture est supérieur à $Tmin$ et inférieur à $Tmax$;

Cohérence pessimiste

Cette stratégie permet d'éviter tout problème de divergence. Elle garantit une cohérence forte. Elle est utilisée quand le taux d'écriture dépasse le seuil $Tmax$.

Les principales fonctions de l'agent de cohérence sont les suivantes :

- Travailler avec différents gestionnaires de cohérence pour faire converger les répliques de la même donnée vers une réplique de référence globale ;
- Diffuser les nouvelles informations de la réplique de référence aux différents nœuds de son site.

A – Traitement d'une requête

Les figures 5.4 et 5.5 mettent en évidence les principales étapes du service de gestion de cohérence pour traiter une requête. Lorsqu'un client soumet une requête, l'agent de cohérence va vérifier s'il s'agit d'une requête de lecture ou d'écriture. Si le type d'accès de la requête est lecture (Figure 5.4), l'agent de cohérence va identifier le niveau de cohérence en fonction du taux d'écriture. Si le niveau de cohérence identifié est cohérence optimiste, alors il va autoriser l'accès immédiatement à n'importe quelle réplique afin de garantir un niveau de performance élevé, mais cette stratégie ne peut pas garantir que toutes les requêtes accèdent à une donnée à jour. Dans le cas où le niveau de cohérence est hybride ; l'agent de cohérence va sélectionner la meilleure réplique disponible et avant d'autoriser l'accès à cette réplique, il va vérifier si la distance de cette réplique est inférieure à la distance tolérée. Si la distance est inférieure, alors l'accès est autorisé à cette réplique. Par contre si la distance est supérieure au seuil, cette réplique est considérée comme une réplique ancienne et dans ce cas l'agent de cohérence va lancer le processus de mise à jour avant d'autoriser l'utilisation de cette réplique. Dans cette stratégie, l'agent de cohérence réalise un compromis entre la cohérence et la performance. Dans le troisième cas où le niveau de cohérence identifié est pessimiste, l'agent de cohérence sélectionne une réplique de référence et autorise l'accès. Si toutes les répliques de référence sont occupées, alors l'agent de cohérence sélectionne la meilleure réplique pour que le nombre de mises à jour soit le plus faible possible et lance le processus de mise à jour ; une fois la mise à jour est terminée, il autorise l'accès.

Dans le cas où le type d'accès de la requête d'un client est une écriture (Figure

5.5), l'agent de cohérence met à jour le taux d'écriture, ensuite il identifie le niveau de cohérence adéquat à l'application. Si le niveau de cohérence est optimiste, Il sélectionne une réplique quelconque et autorise l'accès immédiatement. Cette stratégie ne peut pas garantir que tous les requêtes accèdent à une donnée à jour, mais elle permet de garantir une haute disponibilité. Donc, on peut se trouver avec des répliques très divergentes. En plus, si nous ne contrôlons pas l'accès simultané nous trouverons avec un nombre élevé de situation de conflits même après la propagation de mises à jour. Si le taux d'écritures dépasse le seuil Tmin et reste inférieur à Tmax, alors l'agent de cohérence bascule vers la stratégie de cohérence hybride. Pour traiter une

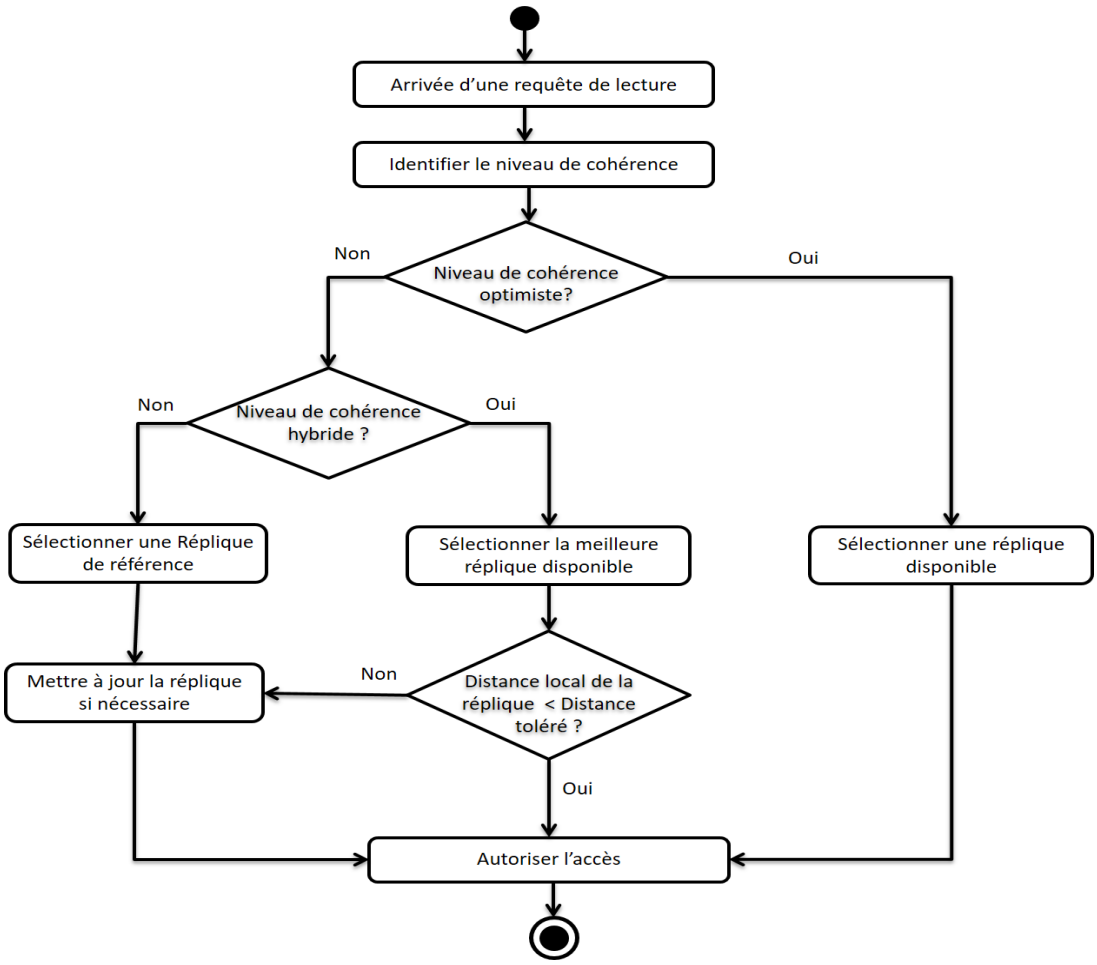


FIG. 5.4 – Traitement d'une requête de lecture

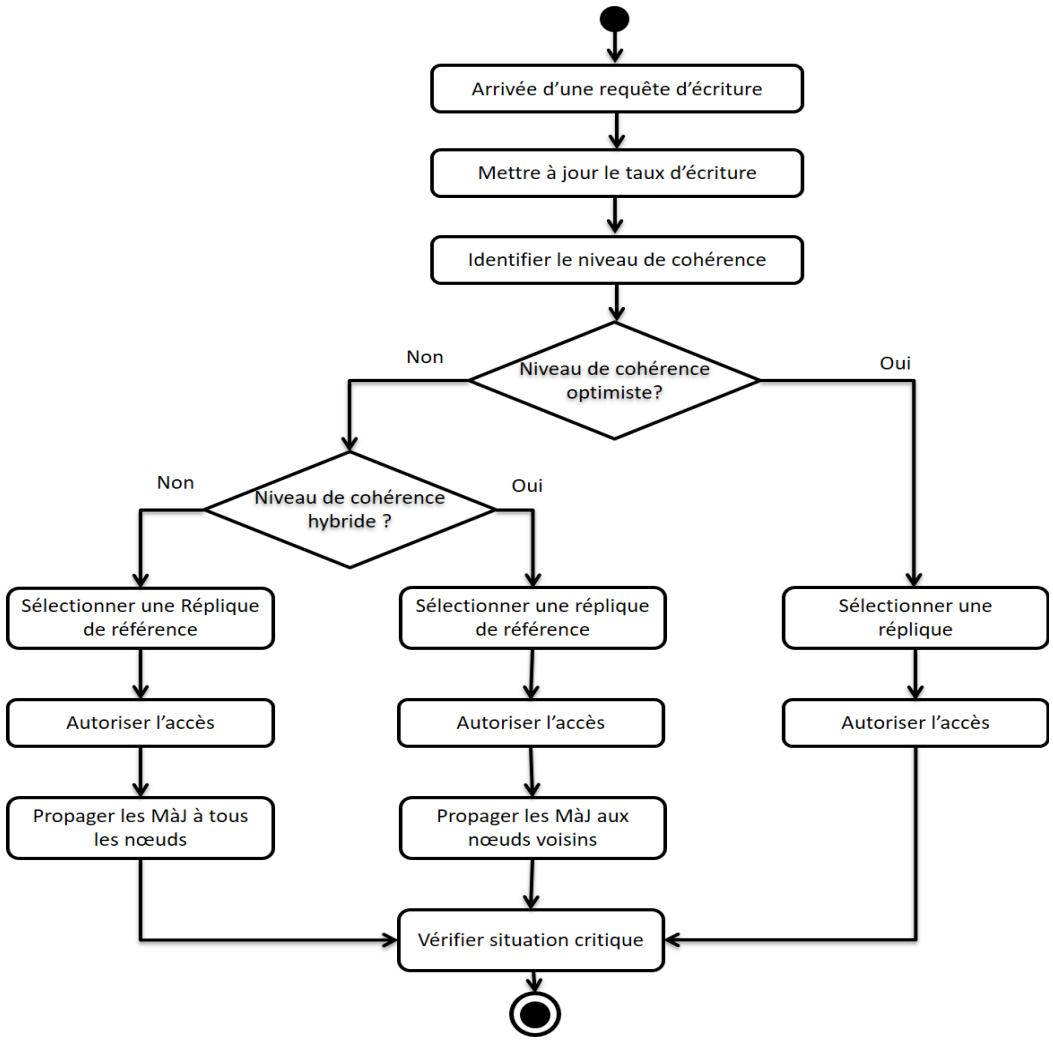


FIG. 5.5 – Traitement d'une requête d'écriture

requête d'un client, l'agent de cohérence sélectionne une réplique de référence locale. Une réplique de référence local c'est une réplique qui a reçu le plus de mise à jour par rapport aux répliques qui se trouvent dans le même centre de donnée. En autorisant l'accès à une réplique de référence local, cela permet de réduire la divergence entre les répliques et minimise le nombre de conflits. Après l'accès à la réplique, l'agent de cohérence propage la mise à jour aux répliques qui se trouvent dans le même centre de donnée seulement. Dans ce cas, l'agent de cohérence réalise un compromis entre cohérence et performance. Quand le taux d'écriture est supérieur à Tmax, l'agent

de cohérence sélectionne le niveau de cohérence pessimiste, donc pour traiter une requête d'un client, l'agent de cohérence sélectionne une réplique de référence et autorise l'accès. Dans cette stratégie, la propagation des mises à jour sera envoyée à tous les répliques du Cloud. Dans ce cas, les données seront moins disponibles afin de ne pas engendrer plus de situation de conflits. Une fois l'accès à la réplique est approuvé, l'agent de cohérence utilise l'algorithme xx pour vérifier s'il y a une situation critique.

5.3.3 Agent de gestion des conflits

L'agent de gestion des conflits est responsable de la coopération et de la négociation entre les autres agents. Les principales fonctions de cet agent sont : détection des conflits, vérification des situations critiques et résolution des conflits.

A – Détection des conflits

Chaque agent de gestion de conflits détecte et résout automatiquement les conflits de réplication lorsqu'ils surviennent. Par exemple, lorsqu'un site transfère sa file d'attente de transactions différées vers un autre site du système, ce dernier site lance automatiquement le processus de détection de conflits entre les répliques. Nous avons utilisé des vecteurs de version pour détecter les conflits dans notre stratégie. Le vecteur de version d'une réplique r est un vecteur de N éléments contenant les numéros de version pour toutes les répliques du système. En supposant que le système contient N répliques du même objet, le vecteur de version d'une réplique r_i est sous la forme $< v_1, v_2, ..., v_n >$, où v_1 signifie la dernière version de r_1, v_2 la dernière version de r_2, et ainsi de suite. Lorsqu'une réplique subit une mise à jour, elle incrémente sa propre entrée dans son vecteur de version.

Pour vérifier l'existence d'un conflit entre deux répliques i et j nous comparons leurs vecteurs de version :

- Si $VV_i = VV_j$, les répliques sont identiques.
- $VV_i \neq VV_j$, si VV_i domine VV_j, alors la réplique du site i est plus récente que la réplique du site j ; c'est-à-dire que le site i a appliqué toutes les mises à jour du site j, et pour maintenir la cohérence des deux répliques, le site j copie le contenu et le vecteur de version (VV) de la réplique du site i. (respectivement, si VV_j domine VV_i alors le contenu et le VV sont copiés de j vers i).

– Dans le cas où $VV_i \neq VV_j$, et ni VV_i domine VV_j, ni VV_j domine VV_i, les opérations sont concurrentes et le système les identifie comme étant en conflit.

B – Situation critique

Un centre de données est en situation critique, si le taux de conflits détecté est supérieur au seuil de conflit toléré ou la distance local dépasse la distance tolérée. Nous utilisons l'algorithme 4 pour vérifier l'existence d'une situation critique dans un centre de données. Dans le cas où une situation critique est trouvée, le service de cohérence local (LCS) entame le processus de négociation. Pour étudier l'évolution de la divergence des répliques à l'intérieur d'un centre de données, nous utilisons les métriques suivantes :

Data: τ^m, D^m
`/* ` τ^m `: Rate of conflicts number tolerated ;`
D^m `: Local distance tolerated */`
initialization;
forall LCS_i **do**
  Calculate τ_i ;
  Calculate D_{local};
  if $(\tau_i > \tau^m)$ *OR* $(D_{local} > D^m)$ **then**
    `/* existence of critical situation of` LCS_i `*/`
    return true;
  else
    return false;
  end
end

Algorithm 4: Vérification d'une situation critique locale

Dlocal

Cette métrique mesure la distance entre les répliques du même objet par rapport à la réplique de référence. Pour calculer la distance, nous construisons un vecteur de référence $VREF_i$ à partir les vecteurs de version des différentes répliques locales,

puis nous calculons la différence entre la composante maximum et la composante minimum du vecteur de référence local. Cette métrique permet de donner une vision sur l'âge des répliques. Cela peut être calculé par :

$$D_{local}(LCS_i) = Max_{t=1}^{n_i}(VREF_i[t]) - Min_{t=1}^{n_i}(VREF_i[t]) \tag{5.1}$$

Taux de conflits (τ_i)

Cette métrique permet de mesurer le taux de conflits pour un centre de données i. Elle permet au service de cohérence local (LCS) de connaître le taux de conflits par rapport au nombre total de répliques d'un même objet dans un centre de données, et elle est donnée par :

$$\tau_i = \frac{ConflictsNbr(LCS_i)}{n_i} \tag{5.2}$$

Où LCS_i est le service de cohérence locale du centre de données i et n_i est le nombre de répliques dans le centre de données i d'un objet donné.

Un LCS détecte une situation critique dans les cas suivants :

$$\begin{gathered} \tau_i > Rate\_of\_conflicts\_number\_tolerated \\ ou \\ Dlocal(LCSi) > Distance\_tolerated \end{gathered} \tag{5.3}$$

C – Résolution de conflit

Une fois qu'un conflit est détecté, il doit être résolu avec l'objectif de la convergence des données sur tous les hôtes. Afin de résoudre une situation de conflit, un processus de négociation entre toutes les parties qui sont en conflit doit avoir lieu. D'abord, nous essayons de résoudre tous les conflits locaux, puis de résoudre le conflit global en utilisant les résultats de la résolution interne des conflits. Le résultat de la négociation devrait être une solution acceptable par tous les agents impliqués. Pour pouvoir utiliser un mécanisme de résolution de conflit dans différents environnements d'exécution, le mécanisme proposé doit consister en plusieurs stratégies de négociation.

Nous allons décrire, dans ce qui suit, le cœur du processus de cohérence basé sur la négociation entre les LCS (voir les algorithmes 5, 6 et 7). Nous commencerons

par donner des définitions de quelques éléments de base, puis nous présenterons le mécanisme de négociation utilisé pour la résolution des conflits entre les répliques.

Tout d'abord, nous choisissons les répliques qui ont subi le plus grand nombre des opérations de mises à jour et nous les ajoutons à un ensemble appelé l'ensemble des répliques candidates (Algorithme 5).

Data: $R = \{r_1, r_2, ..., r_n\}$
/* R : a set of replicas of the same object */
Result: $RC = \{rc_1, rc_2, ..., rc_m\}$
/* RC : a set of candidates' replica */
forall *replica* $r_i \in R$ **do**
  Calculate the average of version vector VV_{r_i};
  /* VV_{r_i} represent version vector of the replica r_i */
  $averVV_i = \frac{\sum_{k=1}^{n} VV_{r_i}[k]}{n}$;
end
/* Search the candidates' replica rc having maximum average of version vectorVV_{r_i} among all replicas $r_i \in R$. */
$RC = \{\max_{i=1}^{n}(averVV_i)\}$;

Algorithm 5: Recherche des répliques candidates

En second lieu, et afin de rechercher la réplique de référence parmi les répliques candidates (Algorithme 6), nous calculons le nombre d'opérations de mise à jour manquantes pour la réplique r_j par rapport à la réplique candidate rc_i. Ensuite, nous calculons le nombre d'opérations de mise à jour qui ont été réalisées dans la réplique r_j et n'a pas été appliqué dans la réplique candidate rc_i.

Enfin, pour sélectionner la réplique de référence (Algorithme 7), nous calculons le nombre d'opérations de mise à jour et de restauration pour la réplique candidate rc_i par rapport à la réplique r_j dans R ; ensuite, nous recherchons la réplique de référence parmi les candidats qui ont le nombre minimum d'opérations de mise à jour et de restauration. La dernière étape consiste à choisir une réplique de référence en utilisant une stratégie de selection.

Data: $R = \{r_1, r_2, ..., r_n\}$, $RC = \{rc_1, rc_2, ..., rc_m\}$
forall *candidate replica* $rc_i \in RC$ **do**
  forall *replica* $r_j \in R$ **do**
    `/* Calculate the number of updates required for each replica` r_j `compared to candidate replica` rc_i `*/`

$$d\_update(rc_i, r_j)_k = \begin{cases} 1 & VV_{rc_i}[k] > VV_{r_j}[k] \\ 0 & else \end{cases} \quad 1 \leq k \leq n;$$

$$D\_UPDATE(rc_i, r_j) = \sum_{k=1}^{n} d\_update(rc_i, r_j)_k;$$

    `/* Calcultate the number of rollback required for each replica` r_j `compared to candidate replica` rc_i `*/`

$$d\_rollback(rc_i, r_j)_k = \begin{cases} 1 & VV_{rc_i}[k] < VV_{r_j}[k] \\ 0 & else \end{cases} \quad 1 \leq k \leq n;$$

$$D\_ROLLBACK(rc_i, r_j) = \sum_{k=1}^{n} d\_rollback(rc_i, r_j)_k;$$

  end
end

Algorithm 6: Recherche des répliques de référence candidates

Dans certaines situations, la propagation des mises ne permet pas de résoudre les conflits entres les répliques ; afin de résoudre ces conflits, il faut annuler certaines mises à jour (roll back) ; mais les opérations d'annulation des mises à jour ont un surcoût sur les performances du système. Pour cela nous essayons de trouver les répliques de références qui permettent de minimiser le nombre d'opération de mises à jour et de restaurations en même temps. Nous utilisons l'lgorithme 7 pour sélectionner la réplique de référence, nous calculons le nombre d'opérations de mise à jour et de restauration pour chaque réplique candidate rc_i par rapport à toutes les répliques r_j dans R ; ensuite, nous recherchons les répliques de références parmi les candidates qui ont le nombre minimum d'opérations de mise à jour et de restauration. Ces répliques sont ajoutées à la liste des répliques de références candidates RRC. La dernière étape consiste à choisir une réplique de référence parmi l'ensemble RRC. Nous favorisons une réplique de référence qui a le nombre minimum de restaurations (roll back) afin

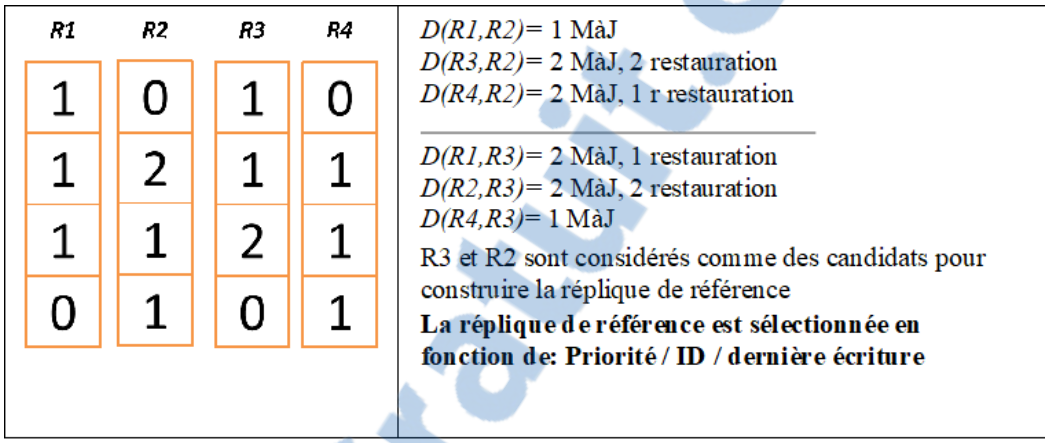
de minimiser le surcoût engendré par les opérations de roll back.

Data: $R = \{r_1, r_2, ..., r_n\}$, $RC = \{rc_1, rc_2, ..., rc_m\}$
Result: rrc_i Reference Replica
forall *candidate replica* $rc_i \in RC$ **do**
  //Calculate the number of total updates and rollback for each candidate replica rc_i ;
  $UPDATE(rc_i) = \sum_{j=1}^{n} D\_UPDATE(rc_i, r_j)$;
  $ROLLBACK(rc_i) = \sum_{j=1}^{n} d\_rollback(rc_i, r_j)$;
end
`/* Find the minimum number of update and rollback operations */`
$Min\_Operation = \min_i^m(UPDATE(rc_i) + ROLLBACK(rc_i))$;
`/* add all replicas that have the minimum operations */`
$RRC = \bigcup_{i=1}^{m} \{rc_i / (UPDATE(rc_i) + ROLLBACK(rc_i)) = Min\_Operation\}$;
if $card(RRC) = 1$ **then**
  `/* ∃ One candidate reference replica` $rrc_i \in RRC$ `*/`
  return rrc_i ;
else
  Let RRC' a set that contain rrc with the minimum of rollback ;
  if $card(RRC') = 1$ **then**
    `/* ∃ One candidate reference replica` $rrc_i \in RRC'$ `*/`
    return rrc_i ;
  else
    use a politic to select the reference replica for example priority/ID/last write ;
    return rrc_i ;
  end
end

Algorithm 7: Sélection de la réplique de référence

D – Exemple de résolution de conflit

Afin de mieux comprendre le mécanisme de résolution de conflit proposé, nous présentons l'exemple suivant.



Exemple de résolution de conflit

Dans cet exemple, nous supposons que nous avons quatre répliques du même objet. La partie gauche de l'exemple présente le vecteur de version de chaque réplique après plusieurs modifications. L'agent de cohérence construit le vecteur de référence en utilisant les vecteurs de version des répliques pour vérifier s'il y a une situation critique. Lorsque l'agent détecte un conflit, il tente de résoudre ce conflit en essayant d'utiliser la stratégie de négociation.

Dans un premier temps, l'algorithme 5 est utilisé pour rechercher les répliques candidates. Dans notre exemple, les répliques R2 et R3 seront considérées comme des répliques candidates puisqu'elles ont subi beaucoup de modifications ; nous cherchons ensuite les répliques de référence candidates en utilisant l'algorithme 6. Nous utilisons maintenant l'algorithme 7 pour sélectionner la réplique de référence ; comme les deux répliques ont le même nombre de mises à jour et de restauration par rapport aux autres répliques. La réplique de référence est sélectionnée en utilisant une des stratégies suivantes :

– Invalider les deux : Suite à cette politique de résolution d'incohérence, les deux versions en conflit sont considérées comme invalides et elles seront restaurées

dans un état cohérent précédent. Cet état est ainsi défini comme l'état cohérent de référence.

- Baser sur l'ID : Dans cette politique, un ID aléatoire est assigné à chaque hôte. Lors de la détection d'un conflit, le système choisit l'hôte qui possède le plus grand ID et considère que sa réplique est une réplique de référence.
- Baser sur la priorité : cette stratégie attribue différentes priorités aux utilisateurs. Lorsqu'un conflit survient, l'utilisateur ayant la priorité la plus élevée gagne et sa réplique est choisie comme référence.

Enfin, les informations (vecteur de version de référence, ensemble des mises à jour) de la réplique de référence sont envoyées à toutes les autres répliques.

5.4 Service de cohérence globale

L'objectif de ce gestionnaire est d'obtenir une cohérence globale, c'est-à-dire de faire converger les répliques stockées dans les différents centres de données vers une réplique de référence globale. En utilisant un algorithme d'élection, un gestionnaire de cohérence local est choisi pour jouer le rôle d'un gestionnaire de cohérence globale. Chaque gestionnaire de cohérence local sélectionne une réplique de référence dans son centre de données. Le vecteur de version de cette réplique de référence locale est envoyé au gestionnaire de cohérence globale. Quand le gestionnaire de cohérence global reçoit les vecteurs de référence de tous les autres centres de donné, il construit le vecteur de référence globale. Ce vecteur de référence globale est utilisé dans le processus de comparaison pour détecter les différences et les conflits entre les différentes répliques de référence. Si des situations de conflit ou de divergences sont détectées, le processus de résolution de conflit et la mise à niveau des répliques seront lancés. Lorsque les situations de divergence ou de conflits sont résolu, le gestionnaire de cohérence global diffuse les nouvelles informations vers les gestionnaires de cohérence local, qui diffusent eux aussi ces nouvelles informations vers leurs répliques locales.

5.5 Conclusion

Dans ce chapitre, nous avons abordé le problème de la cohérence de données. Nous avons proposé une solution de gestion de cohérence flexible et adaptative AC2R pour répondre aux besoins des applications. Au départ, nous avons proposé un modèle

adaptatif et hybride en utilisant les vecteurs de version entre les répliques pour gérer efficacement la cohérence entre les répliques. Dans un deuxième temps, nous avons amélioré la stratégie AC2R proposée par l'intégration d'un mécanisme de convergence et de résolution de conflit entre les répliques qui conviennent au Cloud Computing. Dans le chapitre suivant nous lancerons plusieurs simulations pour examiner le comportement de notre stratégie et nous le comparons avec d'autres stratégies de cohérence.

CHAPITRE 6

Evaluation expérimentale de la stratégie AC2R

6.1 Introduction

Pour évaluer la stratégie AC2R, nous avons étendu le simulateur CloudSim (v3) pour prendre en charge la cohérence des répliques. Les résultats des différentes expérimentations de notre stratégie sont abordés dans ce chapitre. Afin d'analyser les résultats de simulation, nous avons utilisé quatre métriques à savoir le temps de réponse, le nombre de conflits entre les répliques, le coût de communication et la métrique β. Ces métriques ont été utilisées pour évaluer notre stratégie par rapport à l'approche optimiste et à l'approche ROWA.

6.2 Métriques utilisées

Pour évaluer le comportement de la stratégie proposée, nous avons utilisé les métriques suivantes :

Temps de réponse moyen : C'est le temps moyen d'exécution d'une cloudlet (requête), il correspond au temps total d'exécution de toutes les cloudlets divisé par le nombre de cloudlets. Il inclut le temps propre à l'exécution plus le temps de communication dans le réseau (temps d'accès à la donnée) et le temps d'attente.

Nombre de conflits : Un des problèmes majeurs, rencontré dans la phase de

propagation des mises à jour, concerne la détection de conflits. La minimisation de cette métrique est importante pour la gestion de cohérence.

Coût de communication : Quand une réplique subit une modification, il est essentiel de propager les mises à jour vers les autres répliques afin de converger. Nous estimons le coût de communication par le nombre de messages envoyés. Cette métrique indique l'occupation et l'usage du réseau.

Métrique β : Nous avons défini une nouvelle métrique β qui englobe les trois métriques suivantes en même temps : le temps de réponse moyen, le nombre de conflits et le nombre de propagation de mise à jour. Il est évident que la métrique β est à minimiser puisque les trois métriques sont à minimiser aussi. La métrique β permet d'évaluer l'efficacité globale de la stratégie. Dans un premier temps, nous calculons le nombre moyen de toutes les séries de simulations pour chaque métrique. Puis nous calculons le pourcentage de chaque série de simulation par rapport au nombre moyen de chaque métrique. Ensuite nous calculons la moyenne des pourcentages des trois métriques pour chaque série de simulations.

Dans l'exemple suivant, deux stratégies S1 et S2 sont évaluées avec deux métriques : M1 et M2. La valeur $x_{i,j}$ ($y_{i,j}$) représente le résultat obtenu dans la série de simulation j lors de l'évaluation de la stratégie i avec la métrique M1 (M2).

$\alpha_1 = \frac{x_{1,1}+x_{1,2}+x_{2,1}+x_{2,2}}{4} \quad \Longrightarrow \quad \acute{x}_{i,j} = (\frac{x_{i,j}}{\alpha_1})\%$

Métrique M1	Série 1	Série 2
Stratégie S1	$x_{1,1}$	$x_{1,2}$
Stratégie S2	$x_{2,1}$	$x_{2,2}$

Métrique M1 %	Série 1	Série 2
Stratégie S1	$\acute{x}_{1,1}$	$\acute{x}_{1,2}$
Stratégie S2	$\acute{x}_{2,1}$	$\acute{x}_{2,2}$

$\alpha_2 = \frac{y_{1,1}+y_{1,2}+y_{2,1}+y_{2,2}}{4} \Longrightarrow \quad \acute{y}_{i,j} = (\frac{y_{i,j}}{\alpha_2})\%$

Métrique M2	Série 1	Série 2
Stratégie S1	$y_{1,1}$	$y_{1,2}$
Stratégie S2	$y_{2,1}$	$y_{2,2}$

Métrique M2 %	Série 1	Série 2
Stratégie S1	$\acute{y}_{1,1}$	$\acute{y}_{1,2}$
Stratégie S2	$\acute{y}_{2,1}$	$\acute{y}_{2,2}$

Métrique β %	Série 1	Série 2
Stratégie S1	$\frac{\acute{x}_{1,1}+\acute{y}_{1,1}}{2}$	$\frac{\acute{x}_{1,2}+\acute{y}_{1,2}}{2}$
Stratégie S2	$\frac{\acute{x}_{2,1}+\acute{y}_{2,1}}{2}$	$\frac{\acute{x}_{2,2}+\acute{y}_{2,2}}{2}$

6.3 Simulation et résultats

Dans toutes les scénarios de simulation, nous avons utilisé un nombre fixe de répliques qui sont placées dans les hôtes les plus populaires afin de minimiser le temps d'accès aux données et réduire la charge du réseau.

6.3.1 Expérience 1 : Impact du nombre de Cloudlets

Dans cette première série d'expériences, nous évaluons l'impact du nombre de Cloudlet sur différentes métriques. Ces expériences ont été réalisées avec les paramètres de simulation suivants : 2 centre de données, 20 hôtes, 1 donnée et 20 répliques. En ce qui concerne les Cloudlets, nous avons considéré un nombre de Cloudlets qui varie de 20 à 100 par pas de 20. Les résultats de ces expériences sont résumés dans la Figure 6.1, la Figure 6.2 et la Figure 6.3.

A – Effets sur le temps de réponse moyen

La Figure 6.1 montre l'effet du nombre de Cloudlets sur le temps de réponse moyen, nous remarquons que le temps de réponse moyen de l'approche ROWA est nettement grand que les stratégies optimistes et AC2R quel que soit le nombre de

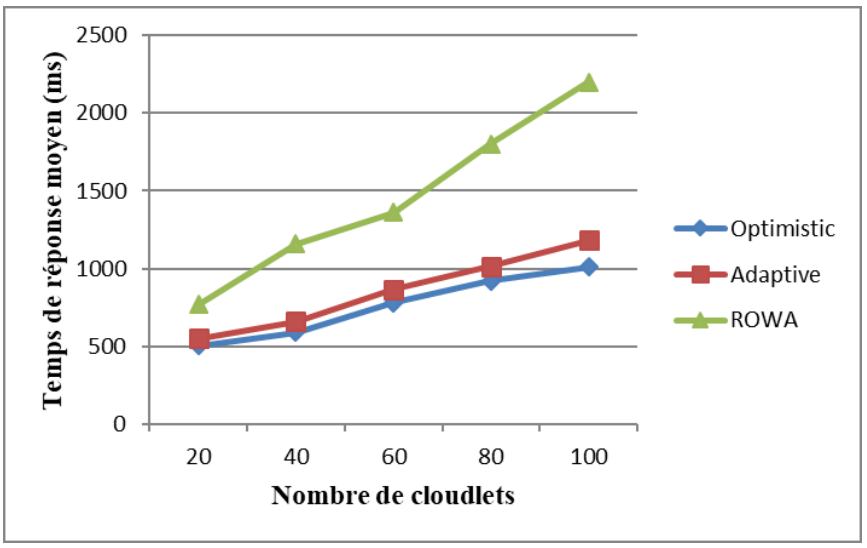


FIG. 6.1 – Variation du temps de réponse moyen par rapport au nombre de cloudlets

Cloudlets exécutés. Ce qui peut être expliqué par le fait que cette approche verrouille toutes les répliques lors d'un accès d'écriture, et donc les autres demandes d'accès seront bloquées jusqu'à la libération des répliques, ce qui a pour effet d'augmenter le temps d'exécution des Cloudlets. Dans la stratégie AC2R l'accès aux répliques est bloqué aussi lors du processus de résolutions de conflits afin de converger les répliques de références locales vers une réplique de référence globale. Les résultats de simulations montrent également que la performance de l'approche optimiste est meilleure que notre stratégie.

B – Effets sur le nombre de conflits

La propagation des mises à jour est essentielle après toute modification afin de faire converger toutes les répliques vers une réplique cohérente. Cependant ces mises à jours peuvent engendrer des situations de conflits. Contrôler et résoudre ceci est essentiel pour la gestion de la cohérence. Deux répliques de la même donnée sont en conflit si la différence entre leurs vecteurs de version comprend plus de deux composantes. Dans la Figure 6.2, nous nous sommes concentrés sur l'évolution du nombre de conflits par rapport au nombre de Cloudlets. L'analyse des résultats de cette Figure montre une diminution du nombre de conflits dans la stratégie proposée.

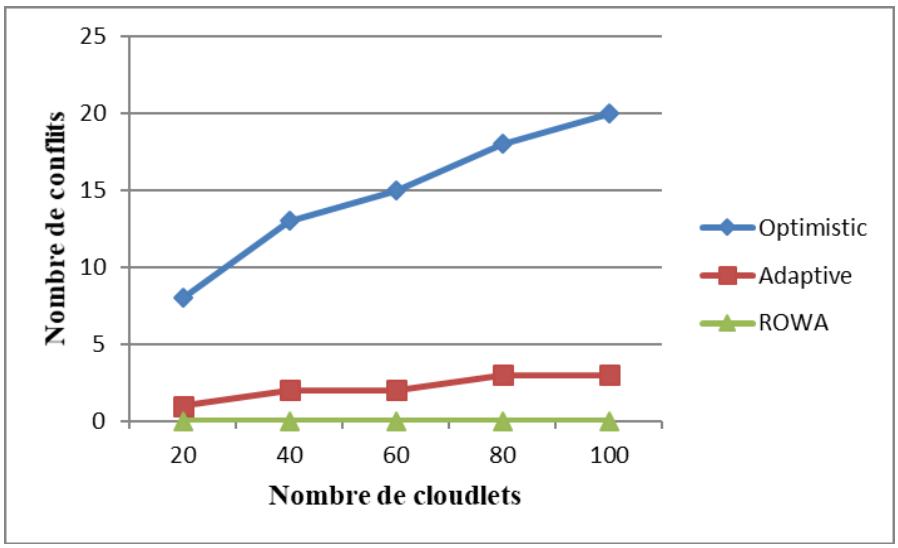


FIG. 6.2 – Variation du nombre de conflits par rapport au nombre de cloudlets

C – Effet sur le coût de communication

Dans l'expérience suivante, nous analysons l'effet du nombre de cloudlets sur le nombre des messages envoyés par les différentes approches. La Figure 6.3 montre le résultat de simulation en termes de nombre de messages de propagation de mise à jour dans le système. Le résultat est exprimé comme un pourcentage par rapport à tous les messages (lecture, mise à jour,...). Nous observons que le nombre de messages de propagation de mise à jour pour ROWA est le plus élevé parmi toutes les approches. En effet, dans ROWA, après chaque modification, une mise à jour est propagée immédiatement à toutes les répliques. Au fur et à mesure que le nombre de Cloudlets augmente, le nombre de messages de propagation de mise à jour augmente aussi. Dans notre stratégie, les mises à jour sont cumulées, puis seront envoyées périodiquement ou afin d'éviter une situation de conflit critique. Le nombre de messages de propagation de mise à jour pour l'approche optimiste est meilleur que la stratégie AC2R lorsque le nombre de Cloudlets est supérieur à 40. les deux stratégies ont presque le même nombre de propagation de MàJ lorsque le nombre de Cloudlets est inférieur à 40.

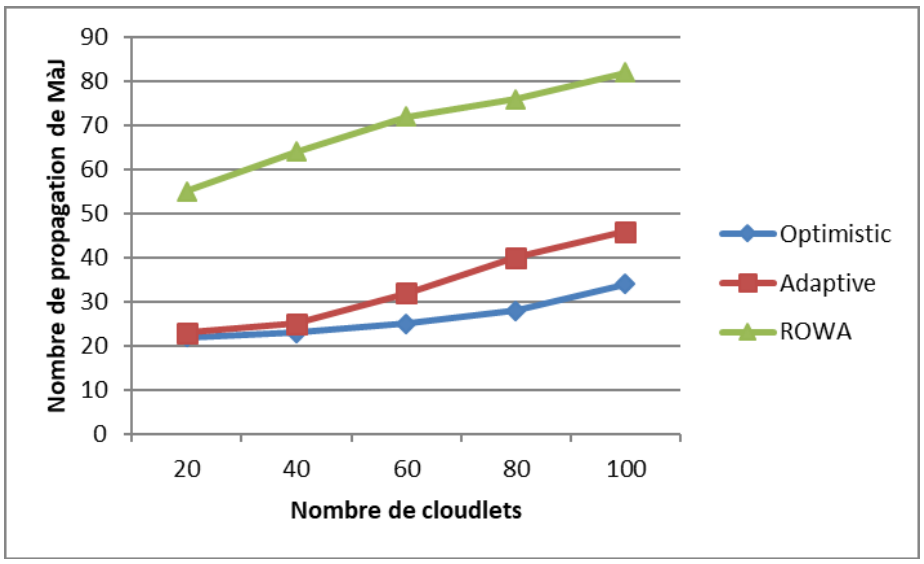


FIG. 6.3 – Variation du nombre de propagations de mise à jour par rapport au nombre de cloudlets

D – Effet sur la métrique β

Dans cette expérience, nous étudions l'impact du nombre de Cloudlets sur la métrique β. Dans la Figure 6.4, nous remarquons que la stratégie ROWA est meilleure par rapport à la stratégie optimiste. Cela peut être expliqué par le nombre de conflits élevé dans la stratégie optimiste qui génère un surcoût sur les performances de systèmes. Nous constatons aussi quel que soit le nombre de Cloudlets, la stratégie AC2R donne des résultats meilleurs par rapport aux autres stratégies. Nous remarquons ainsi un gain de 30% et de 40% respectivement par rapport à la stratégie ROWA et la stratégie optimiste quand le nombre de cloudlet est 20. Le gain d'efficacité de notre stratégie peut atteindre 40% par rapport à la stratégie ROWA et 80 % par rapport à la stratégie optimiste quand le nombre de cloudlets est 100. Ceci est dûe au compromis réalisé par notre stratégie entre la performance, la cohérence et l'utilisation de la bande passante.

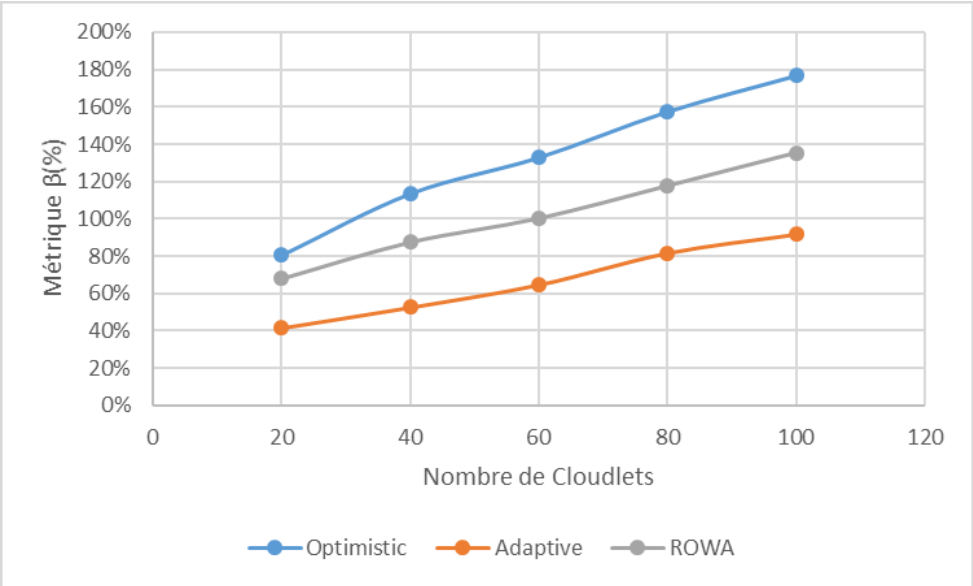


FIG. 6.4 – Effet du nombre de cloudlets sur la métrique β

6.3.2 Expérience 2 : Impact du type de Cloudlet

Dans cette deuxième série d'expériences, nous étudions l'impact du type de Cloudlets sur les différentes métriques. Pour cela, nous avons utilisé les paramètres de si-

mulation suivants : 2 centre de données, 20 hôtes, 1 donnée et 20 répliques. En ce qui concerne les cloudlets, nous avons fixé le nombre de cloudlets à 100 et nous avons varié le type d'accès « écriture » de 20% à 100%. Les résultats de ces expériences sont illustrés dans les figures 6.5 , 6.6 et 6.7.

A – Effets sur le temps de réponse moyen

L'analyse des résultats de la Figure 6.5 montre que lorsque le nombre d'écritures augmente par rapport au nombre de lectures, le temps de réponse moyen augmente. Car une requête d'écriture a besoin de plus de temps d'exécution qu'une requête de lecture, en plus l'approche pessimiste bloque tout le système pour exécuter des Cloudlets de type écriture. Nous notons également que l'approche optimiste est plus efficace que la stratégie AC2R puisque notre stratégie bloque aussi les répliques dans les périodes de convergence ou dans le processus de résolution de conflits.

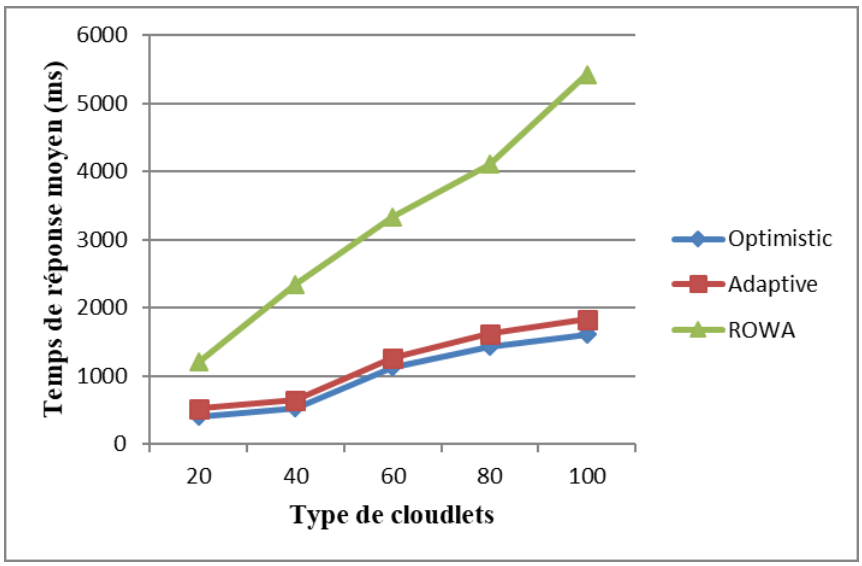


FIG. 6.5 – Variation du temps de réponse moyen par rapport au type de cloudlets

B – Effets sur le nombre de conflits

L'interprétation de la Figure 6.6 montre que la stratégie adaptative pourrait réduire considérablement le nombre de conflits moyen par rapport à l'approche optimiste. On notera également que le nombre de conflits augmente avec l'augmentation de taux d'écriture par rapport au taux de lecture, il est évident que des requêtes

d'écritures entraînent des conflits dûs aux mises à jour effectuées sur l'ensemble des répliques s'il y a des accès concurrents, contrairement à une requête de lecture qui consulte les répliques sans les modifier. Nous observons que la stratégie AC2R permet de réduire le nombre de conflits de 75% par rapport à la stratégie optimiste.

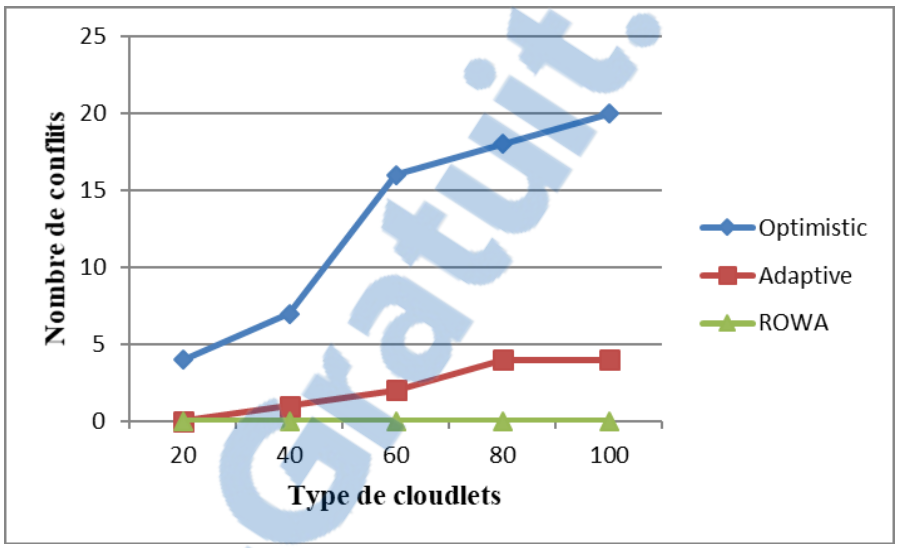


FIG. 6.6 – Variation du nombre de conflits par rapport au type de cloudlets

C – Effet sur le coût de communication

Dans cette expérience, nous comparons le nombre des messages liés à la propagation de mise à jour pour les différentes stratégies. Les paramètres de simulations utilisés sont les même que l'expérience précédente. Nous observons que la stratégie ROWA entraîne un nombre très élevé des messages de mises à jour par rapport aux stratégies adaptative et optimiste (voir la Figure 6.7). Nous remarquons également qu'il n'y a pas de différence significative dans le nombre des messages de mise à jour entre la stratégie AC2R et la stratégie optimiste lorsque le taux d'écriture est inférieur à 40%. La différence augmente à mesure que la valeur du taux d'écriture augmente et elle est maximale lorsque les taux d'écriture atteignent 100%. Afin de réduire le nombre de conflits, et de converger les répliques, notre stratégie effectue plus de mise à jour.

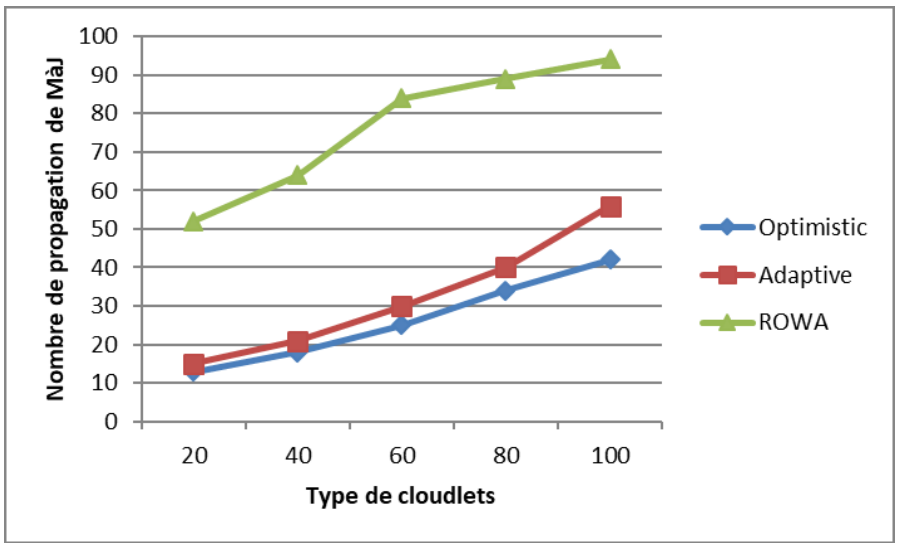


FIG. 6.7 – Variation du nombre de messages par rapport au type de cloudlets

D – Effet sur la métrique β

Dans la dernière expérience, nous analysons l'impact du type des requêtes sur la métrique β. Nous remarquons dans la Figure 6.8 quand le taux d'écriture est

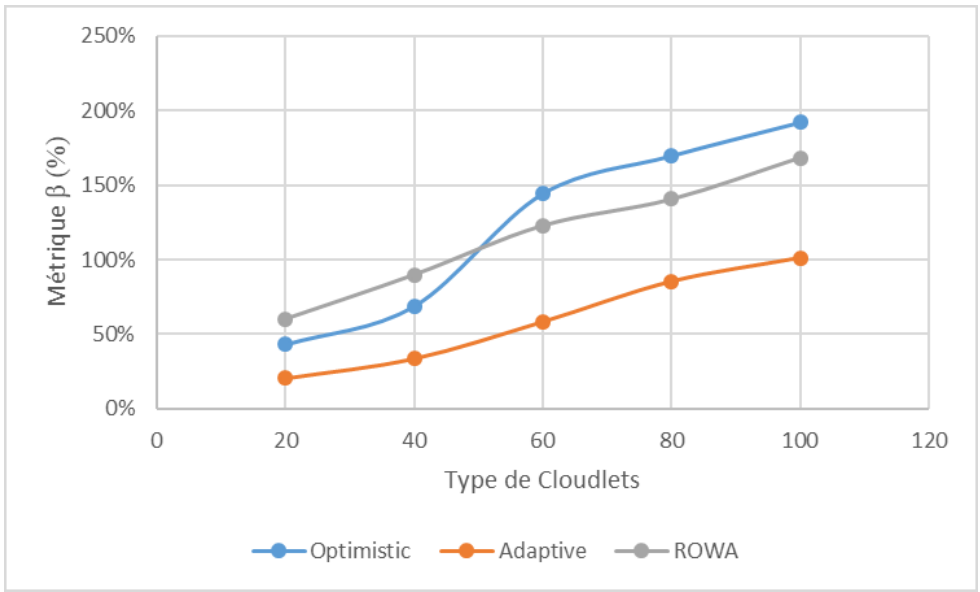


FIG. 6.8 – Effet du nombre de cloudlets sur la métrique β

inférieur à 50%, la stratégie optimiste donne des résultats meilleurs par rapport à la stratégie ROWA. Cette situation s'inverse quand le taux d'écriture dépasse les 50%, qui peut s'expliquer par l'augmentation du nombre de conflits dans la stratégie optimiste. Malgré que le nombre de propagations de mise à jour et le temps de réponse sont meilleurs dans la stratégie optimiste par rapport à la stratégie ROWA. Mais le nombre de conflits élevé dans la stratégie optimiste qui dépasse la moyenne a une influence sur la métrique β. Nous remarquons aussi que la stratégie AC2R est plus efficace par rapport aux autres stratégies quelques soit le taux d'écriture.

6.4 Conclusion

Pour évaluer notre stratégie AC2R de cohérence de données, nous avons lancé une série d'expériences sur notre version étendue du simulateur CloudSim en créant plusieurs scénarios. Les performances sont évaluées à l'aide d'un ensemble de métriques telles que le temps d'exécution, le nombre de conflits et le nombre de propagations de mise à jour. Les résultats obtenus par la simulation montrent l'efficacité de notre stratégie de cohérence en termes de temps d'exécution, en plus, notre stratégie permet le masquage des situations de divergence et d'incompatibilité entre les répliques.

Conclusion générale

Les applications de traitement de données intensives traitent des volumes extrêmement importants. De plus, elles nécessitent des temps de traitement très rapides. La plupart de ces applications sont déployées sur le Cloud pour tirer parti de cette infrastructure, notamment l'évolutivité, la fiabilité et la haute disponibilité, avec un coût réduit. Dans ce contexte, la réplication est un moyen essentiel dans le Cloud. Elle fournit les moyens nécessaires pour garantir la disponibilité des données sur plusieurs copies, l'accès rapide via des copies locales de données, la tolérance aux pannes en répliquant les données sur plusieurs sites.

Nous avons proposé une stratégie de réplication de données qui vise à satisfaire les exigences de disponibilité et de performance tout en maximisant le profit du fournisseur. Afin de préserver les ressources, la réplication est déclenchée uniquement si la disponibilité est inférieure à la disponibilité souhaitée ou si le temps de réponse est supérieur au seuil du temps de réponse, qui sont déjà définis dans le SLA. En outre, la réplication est effectuée uniquement si la création de la nouvelle réplique est rentable pour le fournisseur. Le DRAPP comprend trois points : (i) nous estimons le nombre minimum de réplications requises pour satisfaire les exigences de disponibilité, (ii) pour obtenir les meilleures performances, la planification des requêtes et la réplication des données ont été combinées, et (iii) nous avons optimisé le placement des nouvelles répliques pour améliorer l'équilibrée de charge. Nous avons évalué le

DRAPP en réalisant plusieurs séries d'expériences en variant plusieurs paramètres. Les résultats montrent la supériorité de DRAPP par rapport au CDRM qui est une stratégie déjà proposée pour le Cloud. DRAPP permet de réduire significativement le temps de réponse des requêtes et d'augmenter la disponibilité tout en prenant en compte le profit du fournisseur.

Cependant, la réplication introduit un important problème de cohérence des données. La gestion de la cohérence est primordiale, elle a des conséquences majeures sur les systèmes de stockage distribués. Donc, il est crucial de gérer efficacement la cohérence de données pour répondre aux exigences de performance et de disponibilité dans le Cloud. En raison des latences élevées du réseau, maintenir une cohérence forte est trop coûteux lorsque les systèmes de stockage sont géographiquement distribués. Par conséquent, un modèle de cohérence faible, telle que la cohérence éventuelle, est le choix le plus populaire pour les applications qui requièrent une haute disponibilité et une performances élevée. Cependant, les modèles de cohérence faibles ne sont pas adaptés à toutes les classes d'applications. Plusieurs stratégies ont été proposées pour résoudre ce problème. Ces stratégies utilisent des mécanismes pour changer le niveau de cohérence lors de l'exécution mais la plupart de ces travaux ne montrent pas comment choisir le niveau de cohérence optimal.

En outre, la cohérence éventuelle ne donne aucune garantie sur le moment où toutes les répliques du système de stockage convergent vers un état cohérent. Cependant, l'utilisation de multiples niveaux de cohérence peut impliquer la présence de conflits entre les répliques. La majorité des travaux n'ont pas décrit comment résoudre ce problème, ils ne se contentent que sur la détection des conflits.

Dans notre deuxième contribution, nous avons initialement proposé une stratégie pour assurer la cohérence d'une manière flexible dans les Clouds AC2R. La stratégie proposée ajuste automatiquement le niveau de cohérence lors de l'exécution. Dans un deuxième temps, nous avons abordé le problème de la résolution des conflits entre les répliques ; le mécanisme proposé est basé sur une stratégie de négociation. Ce qui permet de réduire le nombre de conflits et la probabilité des lectures périmées.

Nous avons utilisé la collaboration entre les agents pour tirer parti des différences et des similitudes entre les répliques. Chaque agent de gestion des répliques distribue

les vecteurs de version de sa référence de réplique aux autres agents de gestion de répliques ; le vecteur de version de réplique est utilisé pour construire un vecteur de référence global VREF qui est utilisé par les différents agents de gestion de cohérence pour détecter les convergences, les différences et les conflits entre les répliques ; en présence d'un conflit, les agents de gestion de cohérence collaborent entre eux pour faire converger les répliques de la même donnée vers une réplique de référence globale en utilisant le processus de négociation. Le résultat de la négociation devrait être une solution qui aboutisse à la réalisation des objectifs pour tous les agents impliqués. Pour réaliser la collaboration entre les agents, nous avons utilisé deux types de communication entre eux, la communication globale et la communication locale.

Perspectives

Certaines pistes de recherche sont possibles pour améliorer DRAPP. Nous pouvons projeter de : (i) Étendre les expériences et les comparaisons en étudiant l'influence de la réplication sur la consommation d'énergie et (ii) Valider nos propositions sur une plate-forme Cloud réelle.

Pour les applications qui nécessitent une disponibilité et une performance élevées, la sélection du nombre de répliques et leurs placement affectent ces deux critères. L'un des problèmes de notre stratégie de cohérence AC2R est l'utilisation d'un nombre fixe de répliques et le placement des répliques qui est effectué d'une manière statique. Pour résoudre ce problème, nous utilisons notre première stratégie DRAPP pour calculer le nombre de répliques adéquat et placer ces répliques d'une façon optimale et dynamique. Nous n'avons pas étudié l'impact de la variation du nombre de répliques sur les performances du système ; il serait intéressant d'étudier le comportement de AC2R dans ce scénario.

Un autre problème est envisagé, les deux paramètres : le taux de conflits toléré et la distance locale tolérée qui sont utilisés pour détecter les situations critiques, ne sont pas définis automatiquement.

De plus, la stratégie de négociation peut générer des surcoûts supplémentaires en raison de l'annulation de plusieurs écritures. La proposition d'un algorithme pour définir les deux paramètres en fonction du type de données et des exigences de l'ap-

plication peut augmenter l'efficacité de la stratégie AC2R.

BIBLIOGRAPHIE

[1] Baesens, B., *Analytics in a big data world : The essential guide to data science and its applications*, John Wiley & Sons, 2014.

[2] Beaver, D., Kumar, S., Li, H. C., Sobel, J., and Vajgel, P., "Finding a Needle in Haystack : Facebook's Photo Storage," *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, Vancouver, BC, Canada, 2010, pp. 47–60.

[3] Gantz, J. and Reinsel, D., "The digital universe in 2020 : Big data, bigger digital shadows, and biggest growth in the far east," *IDC iView : IDC Analyze the future*, Vol. 2007, No. 2012, 2012, pp. 1–16.

[4] Dikaiakos, M. D., Katsaros, D., Mehra, P., Pallis, G., and Vakali, A., "Cloud computing : Distributed internet computing for IT and scientific research," *IEEE Internet computing*, Vol. 13, No. 5, 2009, pp. 10–13.

[5] Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., and Brandic, I., "Cloud computing and emerging IT platforms : Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation computer systems*, Vol. 25, No. 6, 2009, pp. 599–616.

[6] Xiong, R., Luo, J., Song, A., Liu, B., and Dong, F., "QoS preference-aware replica selection strategy using MapReduce-based PGA in data grids," *International Conference on Parallel Processing (ICPP)*, IEEE, Taipei City, Taiwan, 2011, pp. 394–403.

[7] Sousa, F. R. and Machado, J. C., "Towards elastic multi-tenant database replication with quality of service," *Fifth International Conference on Utility and Cloud Computing*, IEEE Computer Society, Chicago, IL, USA, 2012, pp. 168–175.

[8] Jin, S. and Wang, L., "Content and service replication strategies in multi-hop wireless mesh networks," *Proceedings of the 8th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*, ACM, Montreal, Quebec, Canada, 2005, pp. 79–86.

[9] Belalem, G. and Slimani, Y., "A hybrid approach to replica management in data grids," *International Journal of Web and Grid Services*, Vol. 3, No. 1, 2007, pp. 2–18.

[10] Ranganathan, K., Iamnitchi, A., and Foster, I., "Improving data availability through dynamic model-driven replication in large peer-to-peer communities," *2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, IEEE, Berlin, Germany, 2002, pp. 376–376.

[11] Tos, U., Mokadem, R., Hameurlain, A., Ayav, T., and Bora, S., "A performance and profit oriented data replication strategy for cloud systems," *Intl IEEE Conferences on Ubiquitous Intelligence and Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld)*, IEEE, Toulouse, France, 2016, pp. 780–787.

[12] Brewer, E. A., "Towards Robust Distributed Systems (Abstract)," *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '00, Portland, Oregon, USA, 2000, pp. 7–.

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

[14] Tabet, K., Mokadem, R., Laouar, M. R., and Eom, S., "Data replication in cloud systems : a survey," *International Journal of Information Systems and Social Change (IJISSC)*, Vol. 8, No. 3, 2017, pp. 17–33.

[15] Kloudas, K., Mamede, M., Preguiça, N., and Rodrigues, R., "Pixida : optimizing data parallel jobs in wide-area data analytics," *Proceedings of the VLDB Endowment*, Vol. 9, No. 2, 2015, pp. 72–83.

[16] Silvestre, G., Monnet, S., Krishnaswamy, R., and Sens, P., "Aren : a popularity aware replication scheme for cloud storage," *18th International Conference on Parallel and Distributed Systems (ICPADS)*, IEEE, Singapore, 2012, pp. 189–196.

[17] Sun, D.-W., Chang, G.-R., Gao, S., Jin, L.-Z., and Wang, X.-W., "Modeling a dynamic data replication strategy to increase system availability in cloud computing environments," *Journal of computer science and technology*, Vol. 27, No. 2, 2012, pp. 256–272.

[18] Wei, Q., Veeravalli, B., Gong, B., Zeng, L., and Feng, D., "CDRM : A cost-effective dynamic replication management scheme for cloud storage cluster," *International Conference on Cluster Computing (CLUSTER)*, IEEE, Heraklion, Crete, Greece, 2010, pp. 188–196.

[19] Belalem, G. and Limam, S., "Towards improving the functioning of CloudSim simulator," *Digital Information Processing and Communications*, Springer, 2011, pp. 258–267.

[20] Qiu, L., Padmanabhan, V. N., and Voelker, G. M., "On the placement of web server replicas," *Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies*, Vol. 3, IEEE, Anchorage, AK, USA, 2001, pp. 1587–1596.

[21] Aazami, A., Ghandeharizadeh, S., and Helmi, T., "Near Optimal Number of Replicas for Continuous Media in Ad-hoc Networks of Wireless Devices," *International Workshop on Multimedia Information Systems*, Salzburg, Austria, 2004, pp. 40–49.

[22] Navimipour, N. J. and Milani, F. S., "A comprehensive study of the resource discovery techniques in peer-to-peer networks," *Peer-to-Peer Networking and Applications*, Vol. 8, No. 3, 2015, pp. 474–492.

[23] Intanagonwiwat, C., Govindan, R., and Estrin, D., "Directed diffusion : A scalable and robust communication paradigm for sensor networks," *Proceedings of the 6th annual international conference on Mobile computing and networking*, ACM, Boston, Massachusetts, USA, 2000, pp. 56–67.

[24] Tang, B., Gupta, H., and Das, S. R., "Benefit-based data caching in ad hoc networks," *IEEE transactions on Mobile Computing*, Vol. 7, No. 3, 2008, pp. 289–304.

[25] Dabrowski, C., "Reliability in grid computing systems," *Concurrency and Computation : Practice and Experience*, Vol. 21, No. 8, 2009, pp. 927–959.

[26] Belalem, G., "A hybrid approach for consistency management in large scale systems," *International conference on Networking and Services ICNS'06*, IEEE, Slicon Valley, CA, USA, 2006, pp. 71–71.

[27] Belalem, G. and Bouhraoua, F., "Dynamic strategy of placement of the replicas in data grid," *International Conference on Parallel Computing Technologies*, Springer, Zalessky, Russia, 2007, pp. 496–506.

[28] Belalem, G. and Limam, S., "Fault tolerant architecture to cloud computing using adaptive checkpoint," *International Journal of Cloud Applications and Computing*, Vol. 1, No. 4, 2011, pp. 60–69.

[29] Belalem, G. and Limam, S., "An approach to fault tolerance in the cloud using the checkpointing technique," *International Journal of Communication Networks and Distributed Systems*, Vol. 11, No. 3, 2013, pp. 236–249.

[30] Limam, S. and Belalem, G., "A migration approach for fault tolerance in cloud computing," *International Journal of Grid and High Performance Computing (IJGHPC)*, Vol. 6, No. 2, 2014, pp. 24–37.

[31] Limam, S. and Belalem, G., "A self-adaptive conflict resolution with flexible consistency guarantee in the cloud computing," *Multiagent and Grid Systems*, Vol. 12, No. 3, 2016, pp. 217–238.

[32] Chervenak, A., Deelman, E., Foster, I., Guy, L., Hoschek, W., Iamnitchi, A., Kesselman, C., Kunszt, P., Ripeanu, M., Schwartzkopf, B., et al., "Giggle : a framework for constructing scalable replica address services," *Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, IEEE Computer Society Press, Baltimore, Maryland, 2002, pp. 1–17.

[33] Čibej, U., Slivnik, B., and Robič, B., "The complexity of static data replication in data grids," *Parallel Computing*, Vol. 31, No. 8-9, 2005, pp. 900–912.

[34] Amjad, T., Sher, M., and Daud, A., "A survey of dynamic replication strategies for improving data availability in data grids," *Future Generation Computer Systems*, Vol. 28, No. 2, 2012, pp. 337–349.

[35] Doğan, A., "A study on performance of dynamic file replication algorithms for real-time file access in data grids," *Future Generation Computer Systems*, Vol. 25, No. 8, 2009, pp. 829–839.

[36] Lee, M.-C., Leu, F.-Y., and Chen, Y.-p., "PFRF : An adaptive data replication algorithm based on star-topology data grids," *Future Generation Computer Systems*, Vol. 28, No. 7, 2012, pp. 1045–1057.

[37] Mokadem, R. and Hameurlain, A., "Data replication strategies with performance objective in data grid systems : a survey," *International Journal of Grid and Utility Computing*, Vol. 6, No. 1, 2014, pp. 30–46.

[38] Tos, U., Mokadem, R., Hameurlain, A., Ayav, T., and Bora, S., "Dynamic replication strategies in data grid systems : a survey," *The Journal of Supercomputing*, Vol. 71, No. 11, 2015, pp. 4116–4140.

[39] Xiong, F., ZHU, X.-x., HAN, J.-y., and WANG, R.-c., "QoS-aware replica placement for data intensive applications," *The Journal of China Universities of Posts and Telecommunications*, Vol. 20, No. 3, 2013, pp. 43–47.

[40] Loukopoulos, T. and Ahmad, I., "Static and adaptive distributed data replication using genetic algorithms," *Journal of Parallel and Distributed Computing*, Vol. 64, No. 11, 2004, pp. 1270–1285.

[41] Chang, R.-S. and Chang, H.-P., "A dynamic data replication strategy using access-weights in data grids," *The Journal of Supercomputing*, Vol. 45, No. 3, 2008, pp. 277–295.

[42] Nicholson, C., Cameron, D. G., Doyle, A., Millar, A. P., and Stockinger, K., "Dynamic data replication in lcg 2008," *Concurrency and Computation : Practice and Experience*, Vol. 20, No. 11, 2008, pp. 1259–1271.

[43] Park, S.-M., Kim, J.-H., Ko, Y.-B., and Yoon, W.-S., "Dynamic data grid replication strategy based on Internet hierarchy," *International Conference on Grid and Cooperative Computing*, Springer, Shanghai, China, 2003, pp. 838–846.

[44] Tang, M., Lee, B.-S., Yeo, C.-K., and Tang, X., "Dynamic replication algorithms for the multi-tier data grid," *Future Generation Computer Systems*, Vol. 21, No. 5, 2005, pp. 775–790.

[45] Sashi, K. and Thanamani, A. S., "A new dynamic replication algorithm for European data grid," *Proceedings of the Third Annual ACM Bangalore Conference*, ACM, Bangalore, India, 2010, p. 17.

[46] Mansouri, N. and Dastghaibyfard, G. H., "A dynamic replica management strategy in data grid," *Journal of network and computer applications*, Vol. 35, No. 4, 2012, pp. 1297–1303.

[47] Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A., and Buyya, R., "CloudSim : a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software : Practice and experience*, Vol. 41, No. 1, 2011, pp. 23–50.

[48] Lei, M. and Vrbsky, S. V., "A Data Replication Strategy to Increase Data Availability in Data Grids," *International Conference on Grid Computing and Applications*, Las Vegas, Nevada, USA, 2006, pp. 221–227.

[49] Chang, R.-S., Huang, N.-Y., Chang, J.-S., et al., "A predictive algorithm for replication optimization in data grids," *Proceeding of ICS 2006*, Taiyuan, Taiwan, 2006, pp. 199–204.

[50] Lin, Y.-F., Wu, J.-J., and Liu, P., "A list-based strategy for optimal replica placement in data grid systems," *37th International Conference on Parallel Processing, 2008. ICPP'08.*, IEEE, Portland, OR, USA, 2008, pp. 198–205.

[51] Bsoul, M., Al-Khasawneh, A., Abdallah, E. E., and Kilani, Y., "Enhanced fast spread replication strategy for data grid," *Journal of Network and Computer Applications*, Vol. 34, No. 2, 2011, pp. 575–580.

[52] Ranganathan, K. and Foster, I., "Identifying dynamic replication strategies for a high-performance data grid," *International Workshop on Grid Computing*, Springer, Denver, CO, USA, 2001, pp. 75–86.

[53] Lei, M., Vrbsky, S. V., and Hong, X., "An on-line replication strategy to increase availability in data grids," *Future Generation Computer Systems*, Vol. 24, No. 2, 2008, pp. 85–98.

[54] Park, S.-M., Kim, J.-H., Ko, Y.-B., and Yoon, W.-S., "Dynamic data grid replication strategy based on Internet hierarchy," *International Conference on Grid and Cooperative Computing*, Springer, Shanghai, China, 2003, pp. 838–846.

[55] Bell, W. H., Cameron, D. G., Carvajal-Schiaffino, R., Millar, A. P., Stockinger, K., and Zini, F., "Evaluation of an economy-based file replication strategy for a data grid," *International Symposium on Cluster Computing and the Grid*, IEEE, Tokyo, Japan, 2003, pp. 661–668.

[56] Lamehamedi, H., Shentu, Z., Szymanski, B., and Deelman, E., "Simulation of dynamic data replication strategies in data grids," *Proceedings International Parallel and Distributed Processing Symposium*, IEEE, Nice, France, 2003, pp. 10–pp.

[57] Gwertzman, J. S. and Seltzer, M., "The case for geographical push-caching," *Proceedings of the Fifth Workshop on Hot Topics in Operating Systems (HotOS-V)*, IEEE, Orcas Island, WA, USA, 1995, pp. 51–55.

[58] Lee, M.-C., Leu, F.-Y., and Chen, Y.-p., "PFRF : An adaptive data replication algorithm based on star-topology data grids," *Future Generation Computer Systems*, Vol. 28, No. 7, 2012, pp. 1045–1057.

[59] Wu, J.-J., Lin, Y.-F., and Liu, P., "Optimal replica placement in hierarchical Data Grids with locality assurance," *Journal of Parallel and Distributed Computing*, Vol. 68, No. 12, 2008, pp. 1517–1538.

[60] Nukarapu, D., Tang, B., Wang, L., and Lu, S., "Data replication in data intensive scientific applications with performance guarantee," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 22, No. 8, 2011, pp. 1299–1306.

[61] Khanli, L. M., Isazadeh, A., and Shishavan, T. N., "PHFS : A dynamic replication method, to decrease access latency in the multi-tier data grid," *Future Generation Computer Systems*, Vol. 27, No. 3, 2011, pp. 233–244.

[62] Horri, A., Sepahvand, R., and Dastghaibyfard, G., "A hierarchical scheduling and replication strategy," *Int J Comput Sci Netw Secur*, Vol. 8, No. 8, 2008, pp. 30–35.

[63] Sidell, J., Aoki, P. M., Sah, A., Staelin, C., Stonebraker, M., and Yu, A., "Data replication in Mariposa," *Proceedings of the Twelfth International Conference on Data Engineering*, IEEE, New Orleans, LA, USA, 1996, pp. 485–494.

[64] Carman, M., Zini, F., Serafini, L., and Stockinger, K., "Towards an economy-based optimisation of file access and replication on a data grid," *International Symposium on Cluster Computing and the Grid*, IEEE, Berlin, Germany, 2002, pp. 340–340.

[65] Lin, H., Abawajy, J. H., and Buyya, R., "Economy-Based Data Replication Broker," *International Conference on e-Science and Grid Computing (e-Science'06)*, Amsterdam, The Netherlands, 2006, p. 90.

[66] Andronikou, V., Mamouras, K., Tserpes, K., Kyriazis, D., and Varvarigou, T., "Dynamic QoS-aware data replication in grid environments based on data "importance"," *Future Generation Computer Systems*, Vol. 28, No. 3, 2012, pp. 544–553.

[67] Zhang, J., Lee, B.-S., Tang, X., and Yeo, C.-K., "A model to predict the optimal performance of the hierarchical data grid," *Future Generation Computer Systems*, Vol. 26, No. 1, 2010, pp. 1–11.

[68] Challal, Z. and Bouabana-Tebibel, T., "A priori replica placement strategy in data grid," *International Conference on Machine and Web Intelligence (ICMWI)*, IEEE, Algiers, Algeria, 2010, pp. 402–406.

[69] Shorfuzzaman, M., Graham, P., and Eskicioglu, R., "Adaptive popularity-driven replica placement in hierarchical data grids," *The Journal of Supercomputing*, Vol. 51, No. 3, 2010, pp. 374–392.

[70] Abdurrab, A. R. and Xie, T., "FIRE : A file reunion based data replication strategy for data grids," *International Conference on Cluster, Cloud and Grid Computing*, IEEE Computer Society, Melbourne, VIC, Australia, 2010, pp. 215–223.

[71] Arlitt, M., Cherkasova, L., Dilley, J., Friedrich, R., and Jin, T., "Evaluating content management techniques for web proxy caches," *ACM SIGMETRICS Performance Evaluation Review*, Vol. 27, No. 4, 2000, pp. 3–11.

[72] Horri, A., Sepahvand, R., and Dastghaibyfard, G., "A hierarchical scheduling and replication strategy," *International Journal of Computer Science and Network Security*, Vol. 8, No. 8, 2008, pp. 30–35.

[73] Mansouri, N. and Dastghaibyfard, G. H., "Enhanced dynamic hierarchical replication and weighted scheduling strategy in data grid," *Journal of Parallel and Distributed Computing*, Vol. 73, No. 4, 2013, pp. 534–543.

[74] Mansouri, N., Dastghaibyfard, G. H., and Mansouri, E., "Combination of data replication and scheduling algorithm for improving data availability in Data Grids," *Journal of Network and Computer Applications*, Vol. 36, No. 2, 2013, pp. 711–722.

[75] Perez, J. M., García-Carballeira, F., Carretero, J., Calderón, A., and Fernández, J., "Branch replication scheme : A new model for data replication in large scale data grids," *Future Generation Computer Systems*, Vol. 26, No. 1, 2010, pp. 12–20.

[76] Zhao, W., Xu, X., Wang, Z., Zhang, Y., and He, S., "A dynamic optimal replication strategy in data grid environment," *International Conference on Internet Technology and Applications*, IEEE, Wuhan, China, 2010, pp. 1–4.

[77] Meroufel, B. and Belalem, G., "Managing data replication and placement based on availability," *AASRI Procedia*, Vol. 5, 2013, pp. 147–155.

[78] Xhafa, F., Kolici, V., Potlog, A.-D., Spaho, E., Barolli, L., and Takizawa, M., "Data replication in P2P collaborative systems," *Proceedings of the 2012 Seventh International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, IEEE, Victoria, BC, Canada, 2012, pp. 49–57.

[79] Abdullah, A., Othman, M., Ibrahim, H., Sulaiman, M. N., and Othman, A. T., "Decentralized replication strategies for P2P based scientific data grid," *International Symposium on Information Technology (ITSim 2008)*, Vol. 3, IEEE, Kuala Lumpur, Malaysia, 2008, pp. 1–8.

[80] Challal, Z. and Bouabana-Tebibel, T., "A priori replica placement strategy in data grid," *International Conference on Machine and Web Intelligence (ICMWI)*, IEEE, Algiers, Algeria, 2010, pp. 402–406.

[81] Chettaoui, H. and Charrada, F. B., "A new decentralized periodic replication strategy for dynamic data grids," *Scalable Computing : Practice and Experience*, Vol. 15, No. 1, 2014, pp. 101–119.

[82] Lamehamedi, H., Szymanski, B., Shentu, Z., and Deelman, E., "Data replication strategies in grid environments," *International Conference on Algorithms and Architectures for Parallel Processing*, IEEE, Beijing, China, 2002, pp. 378–383.

[83] Long, S.-Q., Zhao, Y.-L., and Chen, W., "MORM : A Multi-objective Optimized Replication Management strategy for cloud storage cluster," *Journal of Systems Architecture*, Vol. 60, No. 2, 2014, pp. 234–244.

[84] Gill, N. K. and Singh, S., "Dynamic cost-aware re-replication and rebalancing strategy in cloud system," *International Conference on Frontiers of Intelligent Computing : Theory and Applications (FICTA)*, Springer, Bhubaneswar, India, 2015, pp. 39–47.

[85] Rahman, R. M., Barker, K., and Alhajj, R., "Replica placement design with static optimality and dynamic maintainability," *International Symposium on Cluster Computing and the Grid (CCGRID 06)*, Vol. 1, IEEE, Singapore, 2006, pp. 4–pp.

[86] Ghemawat, S., Gobioff, H., and Leung, S.-T., "The Google File System," *SIGOPS Oper. Syst. Rev.*, Vol. 37, No. 5, 2003, pp. 29–43.

[87] Shvachko, K., Kuang, H., Radia, S., and Chansler, R., "The hadoop distributed file system," *26th Symposium on Mass Storage Systems and Technologies (MSST)*, IEEE, Incline Village, NV, USA, 2010, pp. 1–10.

[88] Li, W., Yang, Y., and Yuan, D., "A novel cost-effective dynamic data replication strategy for reliability in cloud data centres," *International Conference on Dependable, Autonomic and Secure Computing (DASC)*, IEEE, Sydney, NSW, Australia, 2011, pp. 496–502.

[89] Shvachko, K., Kuang, H., Radia, S., and Chansler, R., "The hadoop distributed file system," *International Symposium on Mass Storage Systems and Technologies (MSST)*, IEEE, Incline Village, NV, USA, 2010, pp. 1–10.

[90] Hussein, M.-K. and Mousa, M.-H., "A light-weight data replication for cloud data centers environment," *International Journal of Engineering and Innovative Technology*, Vol. 1, No. 6, 2012, pp. 169–175.

[91] Pamies-Juarez, L., García-López, P., Sánchez-Artigas, M., and Herrera, B., "Towards the design of optimal data redundancy schemes for heterogeneous cloud storage infrastructures," *Computer Networks*, Vol. 55, No. 5, 2011, pp. 1100–1113.

[92] Zhang, H., Lin, B., Liu, Z., and Guo, W., "Data replication placement strategy based on bidding mode for cloud storage cluster," *Web Information System and Application Conference (WISA)*, IEEE, Tianjin, China, 2014, pp. 207–212.

[93] Li, W., Yang, Y., Chen, J., and Yuan, D., "A cost-effective mechanism for cloud data reliability management based on proactive replica checking," *International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, IEEE Computer Society, Ottawa, ON, Canada, 2012, pp. 564–571.

[94] Li, W., Yang, Y., and Yuan, D., "Ensuring cloud data reliability with minimum replication by proactive replica checking," *IEEE Transactions on Computers*, Vol. 65, No. 5, 2016, pp. 1494–1506.

[95] Qu, Y. and Xiong, N., "RFH : A resilient, fault-tolerant and high-efficient replication algorithm for distributed cloud storage," *International Conference on Parallel Processing*, IEEE, Pittsburgh, PA, USA, 2012, pp. 520–529.

[96] Lin, J.-W., Chen, C.-H., and Chang, J. M., "QoS-aware data replication for data-intensive applications in cloud computing systems," *IEEE Transactions on Cloud Computing*, Vol. 1, No. 1, 2013, pp. 101–115.

[97] Lee, J., Chung, J., and Lee, D., "Efficient data replication scheme based on hadoop distributed file system," *Int. J. Softw. Eng. Appl*, Vol. 9, No. 12, 2015, pp. 177–186.

[98] Jayalakshmi, D., TP, R. R., and Srinivasan, R., "Dynamic data replication strategy in cloud environments," *International Conference on Advances in Computing and Communications (ICACC)*, IEEE, Kochi, India, 2015, pp. 102–105.

[99] Bai, X., Jin, H., Liao, X., Shi, X., and Shao, Z., "RTRM : a response time-based replica management strategy for cloud storage system," *International*

Conference on Grid and Pervasive Computing, Springer, Seoul, Korea, 2013, pp. 124–133.

[100] Sakr, S. and Liu, A., "Sla-based and consumer-centric dynamic provisioning for cloud databases," *International Conference on Cloud Computing (CLOUD)*, IEEE, Honolulu, HI, USA, 2012, pp. 360–367.

[101] Zhao, L., Sakr, S., Liu, A., and Bouguettaya, A., "SLA-Driven Database Replication on Virtualized Database Servers," *Cloud Data Management*, Springer, 2014, pp. 97–118.

[102] Sakr, S., Zhao, L., Wada, H., and Liu, A., "Clouddb autoadmin : Towards a truly elastic cloud-based data store," *International Conference on Web Services (ICWS)*, IEEE, Washington, DC, USA, 2011, pp. 732–733.

[103] Agarwal, R., Juve, G., and Deelman, E., "Peer-to-peer data sharing for scientific workflows on amazon ec2," *SC Companion :High Performance Computing, Networking, Storage and Analysis (SCC)*, IEEE, Salt Lake City, UT, USA, 2012, pp. 82–89.

[104] Shen, D., Dong, F., Zhang, J., and Luo, J., "Cost-Effective Virtual Machine Image Replication Management for Cloud Data Centers," *Intl Conf on High Performance Computing and Communications, 6th Intl Symp on Cyberspace Safety and Security, 11th Intl Conf on Embedded Software and Syst (HPCC,CSS,ICESS)*, Paris, France, 2014, pp. 229–236.

[105] Zeng, L., Xu, S., Wang, Y., Cui, X., Kiat, T. W., Bremner, D., and Kent, K. B., "Monetary-and-QoS aware replica placements in cloud-based storage systems," *6th International Conference on Cloud Computing Technology and Science (CloudCom)*, IEEE, Singapore, 2014, pp. 672–675.

[106] Zeng, L., Xu, S., Wang, Y., Kent, K. B., Bremner, D., and Xu, C., "Toward cost-effective replica placements in cloud storage systems with QoS-awareness," *Software : Practice and Experience*, Vol. 47, No. 6, 2017, pp. 813–829.

[107] Huang, M., Ye, X., Wei, S., and Zhu, D., "A strategy of dynamic replica creation in cloud storage," *International Workshop on Cloud Computing and Information Security (CCIS 2013)*, Shanhai,China, 2013, pp. 389–392.

[108] Sharov, A., Shraer, A., Merchant, A., and Stokely, M., "Take me to your leader ! : online optimization of distributed storage configurations," *Proceedings of the VLDB Endowment*, Vol. 8, No. 12, 2015, pp. 1490–1501.

[109] Myint, J. and Naing, T. T., "Management of data replication for PC cluster-based cloud storage system," *International Journal on Cloud Computing : Services and Architecture (IJCCSA)*, Vol. 1, No. 3, 2011, pp. 31–41.

[110] Maheshwari, N., Nanduri, R., and Varma, V., "Dynamic energy efficient data placement and cluster reconfiguration algorithm for mapreduce framework," *Future Generation Computer Systems*, Vol. 28, No. 1, 2012, pp. 119–127.

[111] Liao, B., Yu, J., Sun, H., and Nian, M., "A QoS-aware dynamic data replica deletion strategy for distributed storage systems under cloud computing environments," *International Conference on Cloud and Green Computing (CGC)*, IEEE, Xiangtan, China, 2012, pp. 219–225.

[112] Boru, D., Kliazovich, D., Granelli, F., Bouvry, P., and Zomaya, A. Y., "Energy-efficient data replication in cloud computing datacenters," *Cluster computing*, Vol. 18, No. 1, 2015, pp. 385–402.

[113] Boru, D., Kliazovich, D., Granelli, F., Bouvry, P., and Zomaya, A. Y., "Models for efficient data replication in cloud computing datacenters," *International Conference on Communications (ICC)*, IEEE, London, UK, 2015, pp. 6056–6061.

[114] Tos, U., Mokadem, R., Hameurlain, A., Ayav, T., and Bora, S., "Ensuring performance and provider profit through data replication in cloud systems," *Cluster Computing*, 2017, pp. 1–14.

[115] Tos, U., *Réplication de données dans les systèmes de gestion de données à grande échelle*, Ph.D. thesis, Université de Toulouse, Université Toulouse III-Paul Sabatier, 2017.

[116] Belalem, G. and Yagoubi, B., "Negotiation mechanisms for resolution conflicts among replicas in data grid," *International Journal of Hybrid Information Technology*, Vol. 1, No. 3, 2008, pp. 47–60.

[117] Domenici, A., Donno, F., Pucciani, G., Stockinger, H., and Stockinger, K., "Replica consistency in a data grid," *Nuclear Instruments and Methods in Physics Research Section A : Accelerators, Spectrometers, Detectors and Associated Equipment*, Vol. 534, No. 1-2, 2004, pp. 24–28.

[118] Abawajy, J. H. and Deris, M. M., "Data replication approach with consistency guarantee for data grid," *IEEE Transactions on Computers*, Vol. 63, No. 12, 2014, pp. 2975–2987.

[119] Wada, H., Fekete, A., Zhao, L., Lee, K., and Liu, A., "Data Consistency Properties and the Trade-offs in Commercial Cloud Storage : the Consumers' Perspective," *Conference on Innovative Data Systems Research (CIDR)*, Vol. 11, Asilomar, California, USA, 2011, pp. 134–143.

[120] Golab, W., Rahman, M. R., AuYoung, A., Keeton, K., and Gupta, I., "Client-centric benchmarking of eventual consistency for cloud storage systems," *34th International Conference on Distributed Computing Systems (ICDCS)*, IEEE, Santa Clara, California, USA, 2014, pp. 493–502.

[121] Shen, M., Kshemkalyani, A. D., and Hsu, T.-y., "Causal consistency for geo-replicated cloud storage under partial replication," *International Parallel and Distributed Processing Symposium Workshop (IPDPSW)*, IEEE, Hyderabad, India, 2015, pp. 509–518.

[122] Liu, Q., Wang, G., and Wu, J., "Consistency as a service : Auditing cloud consistency," *IEEE Transactions on Network and Service Management*, Vol. 11, No. 1, 2014, pp. 25–35.

[123] Islam, M. A. and Vrbsky, S. V., "Tree-Based consistency approach for cloud databases," *International Conference on Cloud Computing Technology and Science (CloudCom)*, IEEE, Indianapolis, IN, USA, 2010, pp. 401–404.

[124] Balegas, V., Serra, D., Duarte, S., Ferreira, C., Shapiro, M., Rodrigues, R., and Preguiça, N., "Extending eventually consistent cloud databases for enforcing numeric invariants," *34th Symposium on Reliable Distributed Systems (SRDS)*, IEEE, Montreal, QC, Canada, 2015, pp. 31–36.

[125] Esteves, S., Silva, J., and Veiga, L., "Quality-of-service for consistency of data geo-replication in cloud computing," *Proceedings of the 18th International Conference on Parallel Processing*, Springer, Rhodes Island, Greece, 2012, pp. 285–297.

[126] Wang, X., Yang, S., Wang, S., Niu, X., and Xu, J., "An application-based adaptive replica consistency for cloud storage," *9th International Conference on Grid and Cooperative Computing (GCC)*, IEEE, anjing, China, 2010, pp. 13–17.

[127] Chihoub, H.-E., Ibrahim, S., Antoniu, G., and Perez, M. S., "Harmony : Towards automated self-adaptive consistency in cloud storage," *International Conference on Cluster Computing (CLUSTER)*, IEEE, Beijing, China, 2012, pp. 293–301.

[128] Chihoub, H.-E., "Self-adaptive cost-efficient consistency management in the cloud," *International Symposium on Parallel and Distributed Processing, Workshops and Phd Forum(IPDPSW)*, IEEE, Cambridge, MA, USA, 2013, pp. 2290–2293.

[129] Zhou, Z., Chen, S., Ren, T., and Wu, T., "File heat-based self-adaptive replica consistency strategy for cloud Storage," *Journal of Computers*, Vol. 9, No. 8, 2014, pp. 1928–1933.

[130] Kim, J. and Hwang, S.-T., "Adaptive consistency approaches for cloud computing platform," *International Journal of software engineering and Its applications*, Vol. 9, No. 7, 2015, pp. 127–134.

[131] Elgedawy, I., "DCaaS : Data consistency as a service for managing data uncertainty on the clouds," *International Journal of Advanced Computer Science and Applications(IJACSA)*, Vol. 4, No. 5, 2013, pp. 59–68.

[132] Chen, T., Bahsoon, R., and Tawil, A.-R. H., "Scalable service-oriented replication with flexible consistency guarantee in the cloud," *Information Sciences*, Vol. 264, 2014, pp. 349–370.

[133] Fetai, I. and Schuldt, H., "Cost-based data consistency in a data-as-a-service cloud environment," *5th International Conference on Cloud Computing (CLOUD)*, IEEE, Honolulu, HI, USA, 2012, pp. 526–533.

[134] Chihoub, H.-E., Ibrahim, S., Antoniu, G., and Perez, M. S., "Consistency in the cloud : When money does matter !" *13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, IEEE, Delft, Netherlands, 2013, pp. 352–359.

[135] Terry, D. B., Prabhakaran, V., Kotla, R., Balakrishnan, M., Aguilera, M. K., and Abu-Libdeh, H., "Consistency-based service level agreements for cloud storage," *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, ACM, Farminton, Pennsylvania, 2013, pp. 309–324.

[136] Iskander, M. K., Wilkinson, D. W., Lee, A. J., and Chrysanthis, P. K., "Enforcing policy and data consistency of cloud transactions," *31st International Conference on Distributed Computing Systems Workshops (ICDCSW)*, IEEE, Minneapolis, MN, USA, 2011, pp. 253–262.

[137] Kraska, T., Pang, G., Franklin, M. J., Madden, S., and Fekete, A., "MDCC : Multi-data center consistency," *Proceedings of the 8th ACM European Conference on Computer Systems*, ACM, Prague, Czech Republic, 2013, pp. 113–126.

Liste des travaux de recherche

1. Revues internationales :

a) Belalem, G. and Limam, S., “An approach to fault tolerance in the cloud using the checkpointing technique,” International Journal of Communication Networks and Distributed Systems, Vol. 11, No. 3, 2013, pp. 236–249.
b) Limam, S. and Belalem, G., “A migration approach for fault tolerance in cloud computing,”International Journal of Grid and High Performance Computing (IJGHPC), Vol. 6, No. 2, 2014, pp. 24–37.
c) Limam, S. and Belalem, G., “A self-adaptive conflict resolution with ‡exible consistency guarantee in the cloud computing,” Multiagent and Grid Systems, Vol. 12, No. 3, 2016, pp. 217–238.

2. Conférences internationales :

Limam, S. Mokadem, R. and Belalem, G., "Satisfying Availability and Performance Requirements while Ensuring Profit of Cloud Providers," First international conference on Embedded & Distributed Systems, EDiS'2017, Oran, Algeria, 2017

ملخص

الحوسبة السحابية تجتذب المزيد من الاهتمام لأدائها العالي وتوافرها بتكلفة منخفضة . من بين الطرق التي تتيح توافر واداء عاليين هو استخدام التكرار .وكخطوة أولى، نقترح استراتيجية تكرار ديناميكية تلبي في نفس الوقت احتياجات المستأجرين من حيث التوفر والأداء، مع الأخذ بعين الاعتبار ربح مزود الخدمة .وتستند الاستراتيجية المقترحة على نموذج التكلفة الذي يهدف إلى حساب الحد الأدنى لعدد النسخ المكررة اللازمة للحفاظ على توافر عالي للبيانات .وبالإضافة إلى ذلك، نمزج تكرار البيانات مع جدولة الطلبات لوضع هذه النسخ بطريقة متوازنة، مع الأخذ بعين الاعتبار ميزانية المستأجر .ومن ناحية أخرى، فإن وجود نسخ متماثلة متعددة يدخل مشكلة اتساق البيانات. في الخطوة الثانية، نقترح نموذجا يستند إلى نهج متفائل باستخدام ناقلات الإصدار بين النسخ المتماثلة لإدارة اتساق البيانات .قمنا بتحسين هذه الاستراتيجية من خلال دمج حل النزاع المتماثل وآلية حل النزاعات التي تناسب الحوسبة السحابية.

الكلمات المفتاحية:الأنظمة السحابية، النسخ المتماثل للبيانات، المنافع الاقتصادية، الاتساق، حل النزاعات.

Abstract

As Cloud computing is becoming increasingly popular, Cloud storage services attract more attentions for their high security and availability with a low cost. One way to provide availability and scalability is by the use of replication. Initially, we propose a dynamic replication strategy that satisfies simultaneously availability and performance tenant requirements while taking into account the provider profit. The proposed strategy is based on a cost model that aims to calculate the minimum number of replicas required to maintain a high data availability. Furthermore, data replication and query scheduling are coupled in order to place these replicas in a load balancing way while dealing with the tenant budget. On the other hand, the existence of multiple data replicas introduces a challenge, which is how to maintain replicas consistent. Secondly, we propose a model based on an optimistic approach using versions of vectors between the replicas to manage the consistency of replicas. We improve the proposed strategy by the integration of a convergence mechanism and conflict resolution between the replicas that suits the Clouds Computing.

Keywords: Cloud Systems, Data Replication, Economic Benefit, Consistency, conflict resolution

Résumé

Le Cloud Computing attire plus d'attention pour sa haute performance et sa disponibilité à faible coût. Une façon de fournir la disponibilité et l'évolutivité consiste à utiliser la réplication. Dans un premier temps, nous proposons une stratégie de réplication dynamique qui satisfait simultanément les besoins des locataires en termes de disponibilité et de performances, tout en tenant compte du profit du fournisseur. La stratégie proposée est basée sur un modèle de coût qui vise à calculer le nombre minimum de répliques requises pour maintenir une disponibilité élevée des données. En outre, la réplication des données et l'ordonnancement des requêtes sont couplées afin de placer ces répliques de manière équilibrée en tenant compte du budget du locataire. D'autre part, l'existence de plusieurs répliques introduit le problème de cohérence de données. Dans un deuxième temps, nous proposons un modèle basé sur une approche optimiste utilisant des vecteurs de versions entre les répliques pour gérer la cohérence des données. Nous améliorons cette stratégie par l'intégration d'un mécanisme de convergence et de résolution de conflits entre les répliques qui convient au Cloud Computing.

Mots clés : Cloud Computing, réplication, avantages économiques, cohérence, résolution de conflits

A Migration Approach for Fault Tolerance in Cloud Computing

Said Limam, University of Oran, Es Senia, Oran, Algeria

Ghalem Belalem, University of Oran, Es Senia, Oran, Algeria

ABSTRACT

Cloud computing has become a significant technology and a great solution for providing a flexible, on-demand, and dynamically scalable computing infrastructure for many applications. Cloud computing also presents a significant technology trends. With the cloud computing technology, users use a variety of devices to access programs, storage, and application-development platforms over the Internet, via services offered by cloud computing providers. The probability of failure occur during the execution becomes stronger when the number of node increases; since it is impossible to fully prevent failures, one solution is to implement fault tolerance mechanisms. Fault tolerance has become a major task for computer engineers and software developers because the occurrence of faults increases the cost of using resources. In this paper, the authors have proposed an approach that is a combination of migration and checkpoint mechanism. The checkpoint mechanism minimizes the time lost and reduces the effect of failures on application execution while the migration mechanism guarantee the continuity of application execution and avoid any loss due to hardware failure in a way transparent and efficient. The results obtained by the simulation show the effectiveness of our approaches to fault tolerance in term of execution time and masking effects of failures.

Keywords: *Checkpointing, Cloud Computing, Fault Tolerance, Migration, Virtualization*

INTRODUCTION

Most Cloud computing can be defined as a new style of computing in which dynamically scalable and often virtualized resources are provided as a services over the internet. Cloud computing has become a significant technology trend, and many experts expect that cloud computing will reshape information technology (IT) processes and the IT marketplace. With the cloud computing technology, users use a variety of devices, including PCs, laptops, smartphones, and PDAs to access programs, storage, and application-development platforms over the internet, via services offered by cloud computing providers. It provides the illusion of availability of unlimited resources to fulfill dynamic and variable user requirements. Advantages of the cloud computing technology include cost savings, high availability, and easy scalability (Aljawarneh, 2011; Kaushal & Bala, 2011; Arshad *et al.*, 2012).

Cloud computing takes the technology, services, and applications that are similar to

DOI: 10.4018/ijghpc.2014040102

those on the Internet and turns them into a self-service utility. The use of the word "cloud" makes reference to the two essential concepts:

- **Abstraction:** Cloud computing abstracts the details of system implementation from users and developers. Applications run on physical systems that aren't specified, data is stored in locations that are unknown, administration of systems is outsourced to others, and access by users is ubiquitous;
- **Virtualization:** Cloud computing virtualizes systems by pooling and sharing resources. Systems and storage can be provisioned as needed from a centralized infrastructure, costs are assessed on a metered basis, multi-tenancy is enabled, and resources are scalable with agility (Sosinsky, 2011).

Virtualization techniques are commonly used in cloud platforms to implement partitioning of resources. Instead of having direct access to cloud resources, customers have access to virtual machines, which represent a fraction of a physical machine. Then, we identify three layers in such a cloud infrastructure: the physical resource layer (containing the overall cloud resources), the virtualization layer (containing virtual machines) and the applications layer (containing applications of external companies, which are hosted in the cloud) (Tchana *et al.*, 2012).

The reliability of Cloud computing still remains a major concern among users. Due to economic pressures, these computing infrastructures often use commodity components exposing the hardware to scale and conditions for which it was not originally designed (Vishwanath & Nagappan, 2010). As a result, significantly large numbers of failures manifests in the system and seemingly impose high implications on the hosted applications, impacting their availability and performance. For example, Amazon's Elastic Compute Cloud (EC2) experienced failure in Elastic Block Storage (EBS) drives and network configuration ("Amazon Elastic Compute Cloud,") bringing down thousands of hosted applications and websites for 3 days and 3 hours ("Summary of the Amazon EC2"). Table 1 shows failover records from some of the cloud service provider system.

In this context, applications require fault tolerance abilities so that they can overcome the impact of system failures and perform their functions correctly when failures happen.

Our contribution is to propose an approach of fault tolerance to hide the failure of a component. In the first part we focus on the checkpoint mechanism, which is one of the most used mechanisms in distributed systems. In the second part we consider the combination of backup with migration mechanism.

The rest of this paper is organized as follows. Section 2, we summarize important related works in Cloud Computing. Section 3 presents our fault tolerant approach, Section 4 shows experiments and Section 5 concludes the paper.

RELATED WORK

Cloud computing is expected to be the platform for next generation computing, in which users carry thin clients such as smart phones while storing most of their data in the cloud and submitting computing tasks to the cloud. A web browser serves as the interface between clients and the cloud. Operating system in web browsers allows the users to manage their data and computation tasks. One of the main drivers for the interest in cloud computing is cost and reliability (Deng *et al.*, 2010). Providing highly reliable cloud service is a challenging and critical research problem. It is too expensive to provide redundancy alternative components for all the cloud components, to reduce the cost and to develop highly reliable cloud applications within the limited budget (Bauer & Adams, 2012).

Jing Deng and his co-authors (Deng *et al.*, 2010) propose techniques to improve the fault-tolerance and reliability of a rather general scientific computation: matrix multiplication. Matrix multiplication serves as the foundation for many complex problem solving and opti-

Table 1. Cloud incidents

Organization	Services	Summary	Duration	Date
Dropbox	Website, Mobile Apps, API	Dropbox's website, mobile apps, and API had an outage caused by issues during routine internal maintenance ("Outage post-mortem", 2014)	1 day	2014-01-10
Apple Inc.	iTunes, Apple App Store, iCloud, Calendar	Apple Inc.'s iTunes, App Store, iCloud, and Calendar services had an outage that affected some users ("Apple's App Store, iTunes back online after outages,")	6 hours	2013-02-20
Microsoft Corporation	Windows Azure Compute	Microsoft Windows Azure suffered an extensive, worldwide outage in February that wasn't fully addressed for more than 24 hours ("Microsoft Offers Credits, " 2012)	1 day	2012-02-28
Microsoft Corporation	Microsoft Windows Azure	Microsoft Azure: the service came out of beta. The outage left people without access to their applications ("Microsoft Windows Azure 22-Hour", 2009)	22 hours	2009-03-13
Google, Inc.	Gmail	Users went without Gmail access ("Gmail Back After 30 Hours,")	1 day and 6 hours	2008-10-16

mization. They investigate a cloud selection strategy to decompose the matrix multiplication problem into several tasks which will be submitted to different clouds and they demonstrate that fault-tolerance and reliability against faulty and even malicious clouds in cloud computing can be achieved.

In Goiri *et al.* (2010), authors propose a smart checkpoint for virtualized service providers. It uses Another Union File System to differentiate read-only from read-write parts in the virtual machine image. In this way, read-only parts can be checkpointed only once, while the rest of checkpoints must only save the modifications in read-write parts, thus reducing the time needed to make a checkpoint. The checkpoints are stored in a Hadoop Distributed File System.

For As OpenNebula (Milojicic *et al.* 2011), the Microsoft Windows Azure platform (Windows azure) offers an exclusive FT management at the cloud level ("Microsoft's cloud",). Windows Azure replicates each VM so that a VM failure is covered by its replicas. The Azure solution is limited to web applications developed in the Windows Azure platform ("Amazon Elastic Compute Cloud,"). Moreover, no solution is proposed to repair the failed VM. In addition, for VMs which are not instantiated by the Azure development platform, the entire responsibility of FT management is left to the customer. This is also the case in the Amazon EC2 cloud platform ("Summary of the Amazon EC2").

The authors of the paper (Zheng *et al.*, 2010) propose FTCloud which is a component ranking based framework for building fault-tolerant cloud applications. FTCloud employs the component invocation structures and the invocation frequencies to identify the significant components in a cloud application. An algorithm is proposed to automatically determine optimal fault tolerance strategy for these significant components.

CLEVER addresses FT management, but only for its own components. OpenNebula (Milojicic *et al.* 2011) offers exclusive VM FT implemented at the cloud level. It allows the cloud provider to associate hooks (scripts or programs) with each type of VM failure (ac-

cording to hypervisor information). Hardware failures are not addressed by OpenNebula for two reasons: it provides no hardware sensors and all VM sensors are located on the same machine than the VM. So a machine failure cannot be detected by OpenNebula.

In Kaushal and Bala (2011), the authors propose an FT solution in the cloud at the customer level by replicating servers queries, based on the HA-Proxy load distributor. Others researches such as Zheng (2010) and Slawinska (Slawinska *et al.*, 2010) propose a collaborative solution for specifics applications (MPI for example). However, they do not consider the splitting of the cloud between a (VM) provider and its customers, so their works are only applicable to a SaaS cloud.

Finally, Wenbing Zhao (Zhao *et al.*, 2010) proposes a FT middleware, which can be used by a cloud customer to implement software FT. Their main purpose is to implement a synchronized server replication strategy so that a failed server can be repaired with a consistent state.

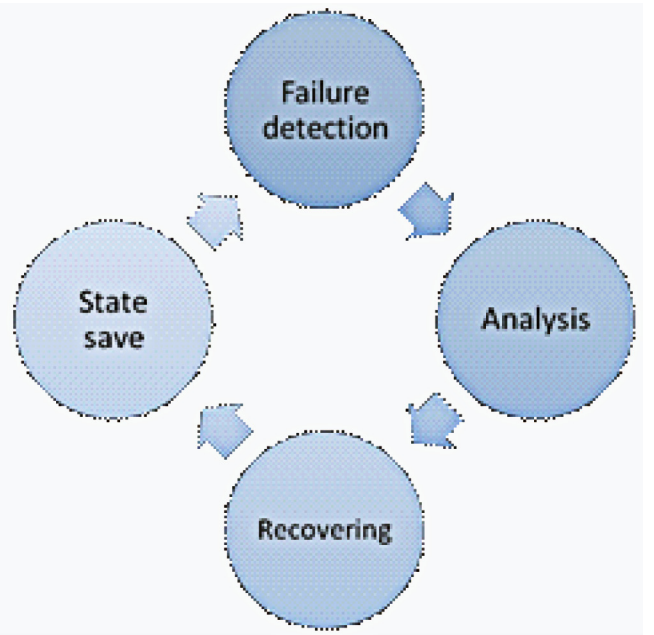
OUR PROPOSED APPROACH

Cloud computing and virtualization have opened a new window in the failure management. Pausing, resuming, and migrating VMs are powerful mechanisms to manage failures in such environments. Virtualization technology is currently becoming increasingly popular and valuable in cloud computing environments due to the benefits of server consolidation, live migration, and resource isolation. Live migration of virtual machines can be used to implement energy saving and load balancing and fault tolerance of systems. A VM can be easily migrated to another node when a failure is predicted or detected.

The main approach of our proposal of fault tolerance is focused on four phases (See Figure 1): backup, error detection, analysis and treatment of errors.

Fault detection and diagnosis are integral parts of any fault tolerance scheme (Hamidi *et al.*, 2012). The detection of errors is the phase

Figure 1. Operational phases



where the presence of a fault is deducted from a failure on a component of the system. We have used a heartbeat mechanism to detect failures: the nodes periodically emit heartbeats or "*I am alive*" messages. If a sufficiently long sequence of heartbeat messages is missed from a node, it is declared to have failed.

Once the failure is detected, this implies a failure of the component and thus stopping the virtual machine. Any damage due to malfunction (failure) is identified in the analysis phase. After these phases, the failure must be corrected and this is done in the phase of error handling.

The structure of our architecture uses a group of agents working in cooperation to control a failure in distributed manner and to initiate the process of fault tolerance.

There are three agents; in Table 2 we describe the role of each agent in the phases of fault tolerance.

The structure FT-Architecture uses a group of processes that collaborate in order to create controller for fault tolerance (See Figure 2).

Checkpoint

In this phase the checkpoint agent save the State of the virtual machine on a stable support to not restart the application from the beginning when failure happens. Several actions of checkpoint are envisaged. Indeed, we can vary the moments and methods of checkpoint. First, we can define a checkpoint mechanism as being periodic; in this case the different checkpoints are performed in a periodic manner. The positions of these checkpoints are fixed and previously determined; we describe in second place another type of checkpoint that we note an Adaptive checkpoint. This mechanism provides greater flexibility in particular at the level of the choice about the moment of checkpoint.

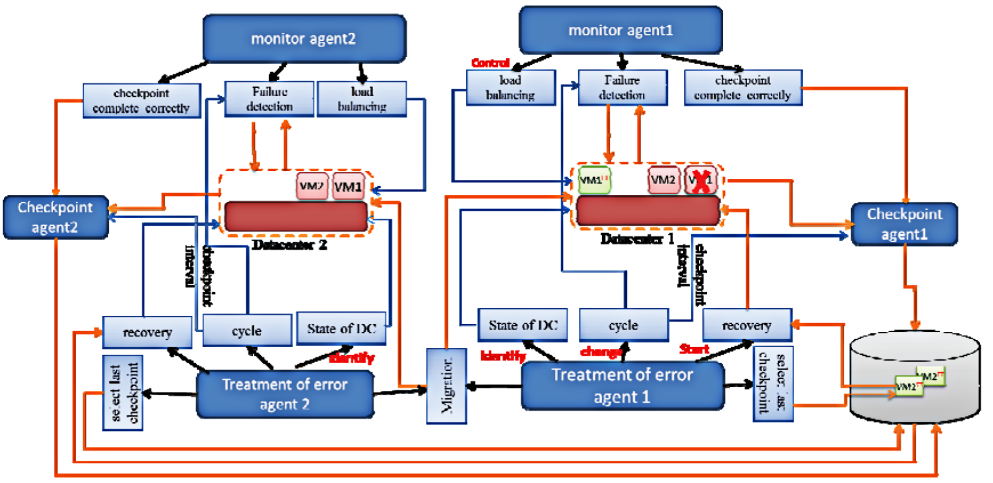
The launch of the checkpoint process increases the execution time of applications: this increase is defined as the overhead of checkpoint. The checkpoint interval also defined the number of checkpoint taken during executions. The interaction between the application and checkpoints determines the extension of the runtime application. Optimal checkpoint interval minimizes the overhead generated by the checkpoints.

To reduce the cost of execution of checkpoint we propose an algorithm which allows to calculate the optimal checkpoint interval based on history, the current state of a node and its execution environment (the State of the Datacenter). By this means, we will, on the one hand, reduce unnecessary checkpoints and, on the other hand, to introduce additional checkpoints, when the danger of failure is considered critical. The algorithm is implemented as shown in Algorithm 1.

Table 2. The role of each element in the phases of fault tolerance

	Element		
Functional Phase	**Checkpoint**	**Treatment of Error Agent**	**Monitor Agent**
State saving	Record the system state periodically	Change the checkpoint interval	Assure that all VMs checkpoint are *completed* correctly
Failure detection		Change the watchdog cycle	Failure detection using a heartbeat watchdog mechanism
Analysis		Identifies the state of the Datacenter	Control the load balancing of Datacenter;
Recovering		Selects the least loaded Datacenter Restart failed processes from their most recent checkpoints	

Figure 2. FT-Architecture proposed



Notations Used

- **Failures:** Number of failures in the last interval;
- **Average of number of failures:** The average of number of failures per interval during the execution;
- **Overhead:** Cost due to the checkpointing and restarting of the applications in the last interval;
- **Waste time:** The total waste time due to F failures in the last interval.

Migration

Migration can improve fault tolerance. If an imminent failure is suspected, the problem can be resolved before the interruption of service. Migration can also be used for load balancing to optimize CPU resource usage and better manage and conserve energy.

Datacenter with a little danger state indicates that it can migrate virtual machines created to other Datacenter. Datacenter with a great danger state indicates that all virtual machines created must be transferred to other data centers. Finally, the state no danger indicates that no effort is useful for transferring virtual machines created (See Figure 3).

When creating a virtual machine, the control element determines the datacenter will run on which the virtual machine. The VM placement policy is described by Algorithm 2.

Local saturation corresponds to the situation where none of the neighbors Datacenter is in *no danger* state. In this situation, one of the neighbors Datacenter is randomly chosen, a high priority is given to the Datacenter which are little danger state. Indeed, the probability of finding a Datacenter in *little danger* state in the vicinity of a Datacenter in a *little danger* state is greater than the in the vicinity of a Datacenter in *critical* state (great danger state). Migration of virtual machines is also used to treat other problems such as fault tolerance and access to resources.

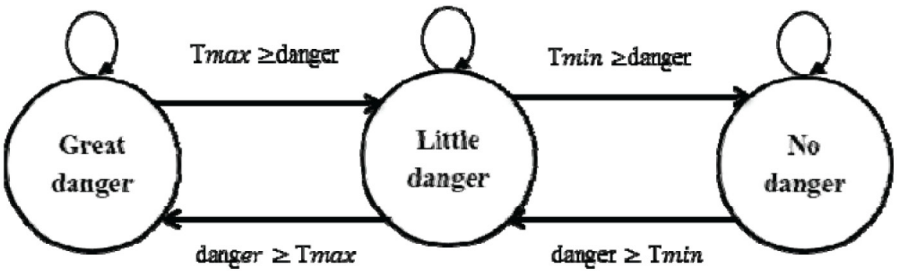
While the adaptive algorithm checkpoint is activated in creating a virtual machine, the migration algorithm is activated at the failure of virtual machines. Indeed, only this event passes a Datacenter in a little danger state. Against failure in multiple virtual machines can switch the state of the Datacenter to the critical state.

Before the error handling agent launches migration mechanism, it executes a search algorithm to choose the future Datacenter. We can consider several search algorithms, to have a good load balancing we used in our approach

Algorithm 1. Calculating the optimal periodic checkpoint interval

```
if (waste time > Overhead) then
    if (number of occupied hosts < 75 %) then
        reduce the checkpoint interval by a factor of 10%;
    else
        reduce the checkpoint interval by a factor of 15%;
    endif
else
    if (failures <= average of number of failures ) then
        if (number of occupied hosts < 50 %) then
            increase the checkpoint interval by a factor of 15%;
        else if (number of occupied hosts < 75 %) then
            increase the checkpoint interval by a factor of 10%;
        else
            increase the checkpoint interval by a factor of 5%;
        endif
    else
        keep the same intervall checkpoint;
    endif
endif
```

Figure 3. Datacenters states



Algorithm 2. VM placement policy

```
if (current state = No danger) then
    create the virtual machine in the current Datacenter;
else if (∃ neighbor/ State(neighbor)=No danger) then
    Create the virtual machine on the neighbor Datacenter;
else
    local saturation;
end
```

an algorithm that selects the least loaded Datacenter. The algorithm is implemented as shown in Algorithm 3.

The research phase of a Datacenter in a no danger implements the following negotiation protocol: the Datacenter in critical state sends a request of migration to the Datacenter in no danger state (having been already selected); this message indicates that the issuer of the motion is in critical state and he asked to make the migration of virtual machines. receiving this message, the Datacenter in no danger state can either acknowledge or reject the request or accept the migration of certain machines (depending on the state of its load) to keep load balancing. A rejection is sent including when the Datacenter is no longer in normal load. During the negotiation phase, the Datacenter in no danger state reserve proposed load. Where the migration query is rejected, the search process continues with other neighboring Datacenters.

Moreover, the time of migration of a virtual machine must not be greater than the time required to complete the task on the current machine. Only tasks with significant execution times can allow a gain by migrating.

SIMULATION AND RESULT

Simulation tools become a critical requirement that not only evaluate the performance of Clouds but also help in developing Cloud computing further (Garg & Buyya, 2011). CloudSim is a generalized and extensible simulation framework that enables seamless modeling, simulation, and experimentation of emerging Cloud computing infrastructures and application services. CloudSim (Buyya *et al.*, 2009) offers the following features: Support modeling and simulation of large scale Cloud computing environments. Support for modeling and simulation of virtualized server hosts, with customizable policies for provisioning host resources to virtual machines. In fact, VM management is the main novelty of this simulator. It also allows simulation of multiple federated data centers to enable studies of VM migration policies for reliability and automatic scaling of applications. However, several aspects of Cloud computing have not been addressed yet such as the simulation of multiple layers simultaneously.

To test our approach we extended CloudSim (version 2.1.1) to support fault tolerance.

The extended CloudSim has the following new features:

- **Fault injection:** This class is used to model the inter-arrival of failures to create scenarios needed to evaluate the performance of our approaches to fault tolerance;
- **Checkpointing and Restarting:** This class models the process of checkpoint/restart, Checkpointing an application is the act of saving the application's state during its execution on stable storage so that if the application fails, it can be restarted from the last saved state;
- **Migrating virtual machines from one datacenter to another.**

We conducted three series of experiments to evaluate our approach to fault tolerance based

Algorithm 3. Choose the future datacenter

```
if (current state = great danger) then
    found ←Search for a Datacenter in State safe;
    if (found) then
        Migrate one (or many) virtual machine from the current Datacenter to
        the selected Datacenter;
    end
end
```

on check pointing and migration. Firstly, the first series of experiments designed to compare the execution time of the combination of migration and checkpoint with checkpoint-based approach and also with an execution without using a fault tolerance protocol. Then we study the impact of failures on the number of virtual machines to migrate. Finally, we study the effect of migration on the load balancing Datacenter.

Effect on the Response Time

Analysis of the results of the Table 3 shows that the two approaches evolve with similar values. We can also notice a big advantage of the first and second approach compared to execution without using a fault tolerance protocol (See Figure 4).

Impact of Failures on the Number of Migration

Through this experience, our goal was to see how increasing the number of failures could have an effect on the number of machines migrated. The results are shown schematically in Figure 5 show that the number of virtual machines to migrate increases with the number of failures.

Effect on the Utilization of Datacenter

This metric shows the percentage of machines used by datacenter, that is to say the number of machines used to create virtual machines divided by the total number of machines is in a datacenter.

Table 3. Effect of frequency of failures on the additional cost of checkpoint

	Execution Time	Waste Time
Without	68316	20316
Checkpoint	51086.7	2611
Migration & checkpoint	51475.9	2369

Figure 4. Execution time for the three approaches

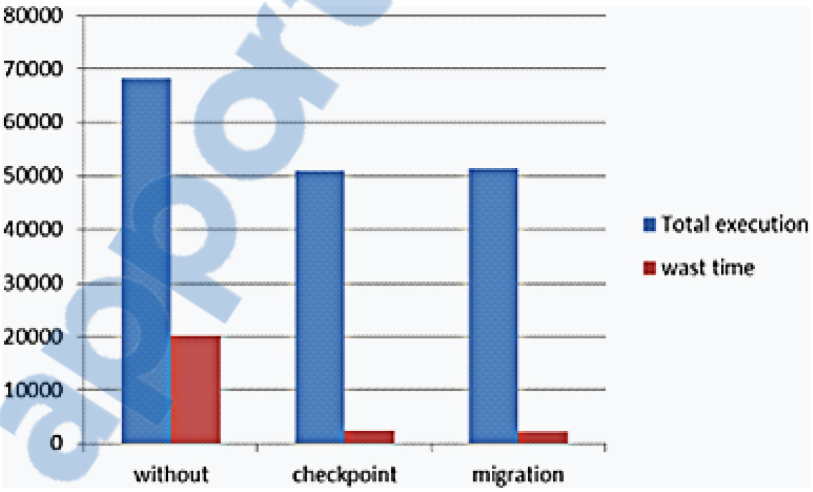
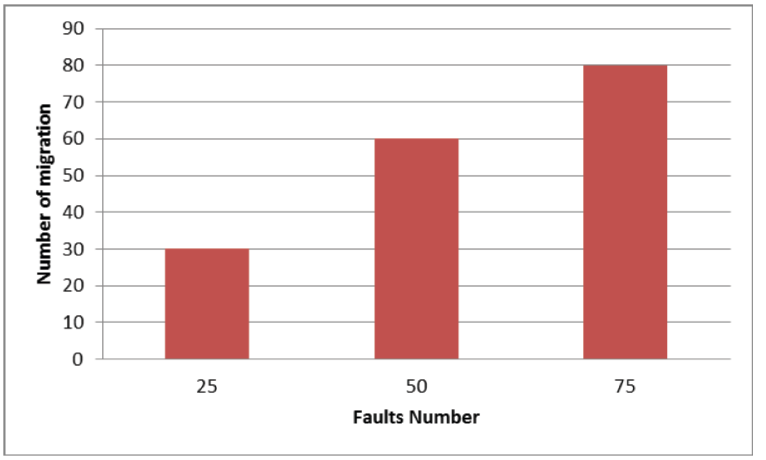


Figure 5. The influence of failures on the number of migration

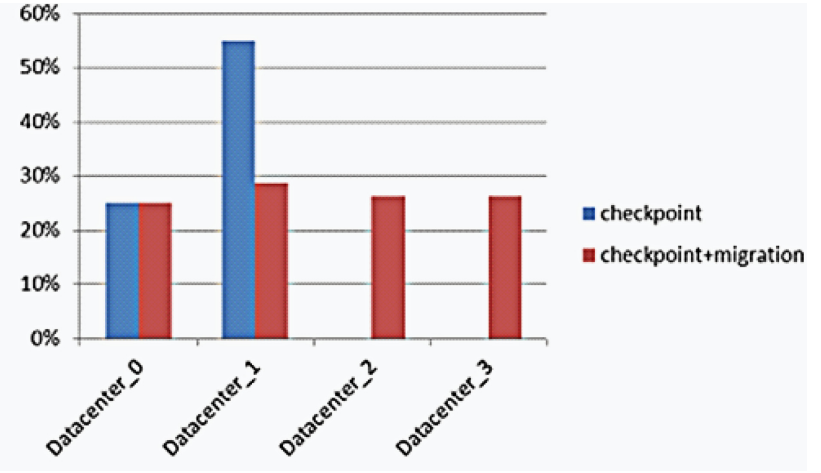


The Table 4 shows the percentage of use of each datacenter by the checkpoint-based approach and the combination of migration and checkpoint approach. The values in this table show a well-balanced use of datacenter with our approach. Figure 6 shows an effective representation of migration on the utilization of datacenter.

Table 4. Effect of migration on the utilization of datacenter

	Datacenter_0	Datacenter_1	Datacenter_2	Datacenter_3
Checkpoint	25%	55.00%	0	0
Checkpoint & Migration	25%	28.75%	26.25%	26.25%

Figure 6. Effect of migration on the utilization of datacenter



Synthesis Experiments

The Table 5 compares the execution time of the two approaches proposed fault tolerance. The column "Time Gained" represents the time gained on the processing time for applications by implementing the first and the second approach with the presence of failures with respect to the processing time without using a fault tolerance mechanism. Read the values in this table clearly illustrate the decrease in total execution time when we use a fault tolerance protocol. The maximum execution time with the approach based on the checkpoint (103,400 seconds, with a failure rate of 75%) is 24.02% smaller than the execution time without using a mechanism for fault tolerance (136,090 seconds) and 1.81% smaller than the execution time with an approach based on the combination of checkpoint and migration (105,856 seconds).

We note that when the checkpoint interval is smaller, overhead are caused by the checkpoint a little greater than when the checkpoint interval is large (See Figure 7). The reason for this is the number of checkpoint during application execution. On the other hand when the checkpoint interval is larger, the loss of computation time caused by failures is greater.

The computation time lost caused by failure depends entirely on when this failure to occur. For example, a failure that occurs after a few seconds the backup does not cause the loss of a few seconds of computation time. If, however, this failure occurs a few seconds before starting a backup, the computation time lost may be important and advantage of the approach based on adaptive checkpoint becomes increasingly clear in these cases. In the approach based on adaptive checkpoint, a moment checkpoint depends on the history (history of failures) and the current state of a node and its execution environment, which reduces the time lost due to overhead backup and lost work after failures. The second approach allows a better distribution of load to ensure a high quality level and permanent application. On the one hand, this solution overcomes the failure of a datacenter and it has almost the same results as the first approach point of view of execution time. On the other hand, it allows multiple Datacenter to improve application performance and ensure the best response time.

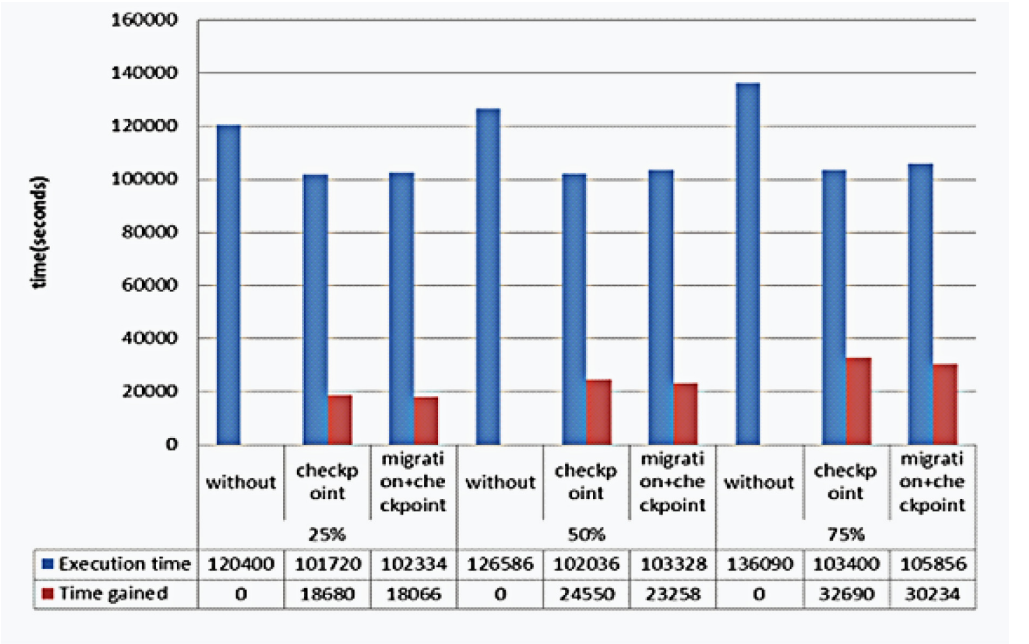
CONCLUSION AND FUTURE WORK

One of the most important for a more efficient use of cloud computing is undoubtedly the study of the reliability and robustness of these systems. In such a context, the safety of applications is a critical element in which the

Table 5. The overall execution time and the time gained in three approaches

Faults Numbers	Approach	Execution Time	Time Gained	% of Time Gained
25	Without	120400	0	0
	Checkpoint	101720	18680	15.51%
	Checkpoint and Migration	102334	18066	15.00%
50	Without	126586	0	0
	Checkpoint	102036	24550	19.39%
	Checkpoint and Migration	103328	23258	18.37%
75	Without	136090	0	0
	Checkpoint	103400	32690	24.02%
	Checkpoint and Migration	105856	30234	22.21%

Figure 7. The overall execution time and the time gained in three approaches



failure rate in such systems increases with the size of the system itself, which is why a fault tolerance mechanism becomes necessary to ensure some aspects.

Fail inevitably; an application will fail, no matter what its environment. In this scenario, users and system administrators will need tools and mechanisms that help them to manage failures and with as little influence on the execution of application.

We have launched a series of experiments by creating multiple failure scenarios. The performances are evaluated using a set of metrics such as number of failed cloudlets, the execution time, the overhead of checkpoint, the overhead of migration and lost time caused by failures. The results show that the proposed approach has good performance, it induces in particular a low overhead on the execution time and they minimize lost time caused by failures.

The research work that we conducted in the context of the issue of fault tolerance in the cloud, as well as the results we obtained showed that our proposals have provided actually better results than the approach without fault tolerance.

However, some research tracks are possible to continue this work. Among these tracks of research we can mention the following:

- Validation of our proposals on a cloud computing platform, such as: Eucalyptus, Nebula;
- Exploit replication to improve performance (Meroufel & Belalem, 2014);
- Use incremental backup, its principle is to back up only the data that has been modified since the last backup. This reduces the overhead of checkpoint and the volume of data to back up because all data may not be altered between each step backup;
- Extend experiments and comparisons that we made in studying the influence of migration on the energy consumption of data center.

REFERENCES

Aljawarneh, S. (2011). Cloud security engineering: Avoiding security threats the right way. [IJCAC]. *International Journal of Cloud Applications and Computing*, *1*(2), 64–70. doi:10.4018/ijcac.2011040105

Amazon Elastic Compute Cloud. (n.d.). Retrieved from http://aws.amazon.com/ec2/#pricing

Apple's App Store. (n.d.). *iTunes back online after outages*. Retrieved from http://news.cnet.com/8301-13579_3-57570697-37/apples-app-store-itunes-back-online-after-outages/

Arshad, J., Townend, P., Xu, J., & Jie, W. (2012). Cloud computing security: Opportunities and pitfalls. [IJGHPC]. *International Journal of Grid and High Performance Computing*, *4*(1), 52–66. doi:10.4018/jghpc.2012010104

Bauer, E., & Adams, R. (2012). Reliability and availability of cloud computing. In J. Anderson (Ed.). Wiley-IEEE Press.

Buyya, R., Ranjan, R., & Calheiros, R. N. (2009, June 21-24). Modeling and simulation of scalable cloud computing environments and the CloudSim toolkit: Challenges and opportunities. In *International the High Performance Computing & Simulation (HPCS'09)*, Leipzig, Germany (pp. 1–11).

Deng, J., Huang, S. C.-H., Han, Y. S., & Deng, J. H. (2010). Fault-tolerant and reliable computation in cloud computing. In *Proceedings of IEEE Globecom 2010 Workshop on Web and Pervasive Security (WPS 2010)*, Miami, FL. doi:10.1109/GLOCOMW.2010.5700210

Garg, S. K., & Buyya, R. (2011). *Network CloudSim: Modelling parallel applications in cloud simulations*. Paper presented at the 4th IEEE International Conference on Utility and Cloud Computing (UCC 2011), Melbourne, Australia. doi:10.1109/UCC.2011.24

Goiri, I., Julià, F., Guitart, J., & Torres, J. (2010, April 19-23). Checkpoint-based fault-tolerant infrastructure for virtualized service providers. In *Proceedings of 12th IEEE/IFIP Network Operations and Management Symposium (NOMS'10)*, Osaka, Japan (pp. 455-462). doi:10.1109/NOMS.2010.5488493

Hamidi, H., Vafaei, A., & Monadjemi, S. A. H. (2012). Analysis and evaluation of a new algorithm based fault tolerance for computing systems. [IJGHPC]. *International Journal of Grid and High Performance Computing*, *4*(1), 37–51. doi:10.4018/jghpc.2012010103

Kaushal, V., & Bala, A. (2011). Autonomic fault tolerance using HAProxy in cloud environment. *International Journal of Advanced Engineering Sciences and Technologies*, *7*(2), 222–227.

Meroufel, B., & Belalem, G. (2014). Collaborative services for fault tolerance in hierarchical data grid. [IJDST]. *International Journal of Distributed Systems and Technologies*, *5*(1), 1–21. doi:10.4018/ijdst.2014010101

Microsoft. (n.d.). *Microsoft's cloud services platform*. Retrieved from http://www.microsoft.com/windowsazure/

Microsoft offers credits for 'Leap Day' Azure outage. (2012). Retrieved March 29, 2012, from http://www.pcworld.com/article/251693/microsoft_offers_credits_for_leap_day_azure_outage.html

Microsoft Windows Azure 22-hour outage takes down weekend users. (2009). Retrieved December 10, 2009, from www.eweek.com/c/a/Windows/Microsoft-Windows-Azure-22Hour-Outage-Takes-Down-Weekend-Users-175615/

Milojicic, D. S., Llorente, I. M., & Montero, R. S. (2011). OpenNebula: A cloud management tool. *IEEE Internet Computing*, *15*(2), 11–14. doi:10.1109/MIC.2011.44

Outage post-mortem. (2014). Retrieved from https://tech.dropbox.com/2014/01/outage-post-mortem/

PcWorld.com. (n.d.). *Gmail back after 30 hours down*. Retrieved January 20, 2012, from http://www.pcworld.com/article/152407/gmail_back_after_30_hours_down.html

Slawinska, M., Slawinski, J., & Sunderam, V. (2010, April 19-23). Unibus: Aspects of heterogeneity and fault tolerance in cloud computing. In *Proceedings 24th IEEE International Symposium on Parallel and Distributed Processing (IPDPS'2010)*, Atlanta, GA (pp. 1-10). doi:10.1109/IPDPSW.2010.5470876

Sosinsky, B. (2011). *Cloud computing bible*. Wiley, John & Sons Incorporated.

Summary of the Amazon EC2 and Amazon RDS service disruption in the US East Region. (2011). Retrieved April 29, 2011, from http://aws.amazon.com/message/65648

Tchana, A., Broto, L., & Hagimont, D. (2012, May 14-16). Approaches to cloud computing fault tolerance. In *Proceedings of the International Conference on Computer, Information and Telecommunication Systems*, Amman, Jordan (pp. 1-6).

Vishwanath, K. V., & Nagappan, N. (2010). Characterizing cloud computing hardware reliability. In *Proceedings of the 1st ACM symposium on Cloud computing, Ser. (SoCC'10)*, New York, NY (pp. 193–204). doi:10.1145/1807128.1807161

Zhao, W., Melliar-Smith, P. M., & Moser, L. E. (2010). Fault tolerance middleware for cloud computing. In *Proceedings of the 3rd International Conference on Cloud Computing (CLOUD'2010)*, Miami, FL (pp. 67-74). doi:10.1109/CLOUD.2010.26

Zheng, Q. (2010, May 19-23). Improving mapreduce fault tolerance in the cloud. In *Proceedings of the International Symposium on Parallel and Distributed Processing, Workshops and Phd Forum (IPDPSW'2010)*, Atlanta, GA (pp. 1-6).

Zheng, Z., Zhou, T. C., Lyu, M. R., & King, I. (2010). FTCloud: A component ranking framework for fault-tolerant cloud applications. In *Proceedings of the 21th IEEE International Symposium on Software Reliability Engineering (ISSRE 2010)* (pp. 398-407). doi:10.1109/ISSRE.2010.28

Said Limam is a PhD candidate in the Department of Computer Science in the Faculty of Sciences at the University of Oran in Algeria. He received his M.S. degree in 2012 from the University of Oran. His research interests are: distributed system, grid computing, cloud computing, fault tolerance, data replication and consistency.

Ghalem Belalem Graduated from department of computer science, Faculty of Sciences, University of Oran, Algeria, where he received PhD degree in computer science in 2007. He is now a research fellow of management of replicas in data replicas in data grid. His current research interests are distributed system; grid computing, could computing and data grid placement of replicas, consistency, fault tolerance, economic models, energy, Big data, and improved performance in large scale systems and mobile environment.

Multiagent and Grid Systems – An International Journal 12 (2016) 217–238
DOI 10.3233/MGS-160251
IOS Press

A self-adaptive conflict resolution with flexible consistency guarantee in the cloud computing

Said Limam* and Ghalem Belalem
Department of Computer Science, Faculty of Exact and Applied Sciences, University of Oran, Ahmed Ben Bella, Algeria

Received 25 February 2016
Accepted 8 July 2016

Abstract. As cloud computing is becoming increasingly popular, cloud storage services attract more attentions for their high security and availability with a low cost. One way to provide availability and scalability is by the use of replication, which places multiple copies of the data in different servers in order to survive to server failures (availability) and to distribute the request load (scalability). On the other hand, the existence of multiple data replicas introduces a challenge which is how to maintain replicas consistent. Initially we proposed in this paper a model based on an optimistic approach using versions of vectors between the replicas to manage the consistency of replicas. In a second step we improved the approach proposed by the integration of a convergence mechanism and conflict resolution between the replicas that suits the Clouds Computing. This mechanism is based on the different forms of negotiation between Local Consistency Service (LCS). We made the modeling and formulation of the proposed model. We also evaluated the performance of this model. The results obtained by the simulation show the effectiveness of our approach in term of execution time and masking divergence and incompatibility situations.

Keywords: Consistency, replica, conflict resolution, cloud computing, system availability, CloudSim

1. Introduction

Cloud computing is a natural extension of many of the design principles, protocols, plumbing, and systems that have been developed over the past 20 years. Such as grid computing, parallel computing, but cloud computing has its own characteristics that some authors [1–3] indicated by drawing detailed comparisons with other traditional IT infrastructure and similar technologies.

As cloud computing is becoming increasingly popular, cloud storage services attract more attentions for their high security and availability with a low cost. Cloud storage is expected to become the main force of the future storage market.

For cloud computing to be a real alternative to grid and cluster computing, it should perform well with everyday applications. Nowadays many of these applications are data-intensive: companies like Google, Amazon, and Facebook deal with peta- and terabytes of data every day. Such corporations

*Corresponding author: Said Limam, Department of Computer Science, Faculty of Exact and Applied Sciences, University of Oran 1, BP. 1524, EL M'Naouer, Oran, Algeria. E-mail: said.limam@gmail.com.

ISSN 1574-1702/16/$35.00

typically have large infrastructures that allow them to cope with the data tsunami. On the other hand, few organizations can acquire the resources and infrastructure necessary to act face the deluge of data and small companies remained unequipped. This Data Deluge phenomenon, introduces many challenges and problems. Therefore, experts deal with issues that relate to how to store, manage, process and query data on a daily basis [4]. The continuous growth rate of data size and variety demonstrates the necessity for innovative and efficient ways in order to address the emerging challenges of Big Data. In this context, organizations should expect to handle and process overwhelming amounts of data that exceed their capacities.

Storage management and performance within clouds is extremely important. Storage systems often rely on replication; it is a strategy in which multiple copies of data, programs, processes, physical property, etc., are stored on several nodes [5].

Data replication (duplication of data across multiple nodes), as shown in the following figure is an effective solution to achieve good performance and better data availability. The main goal of replication is to facilitate access to data, while increasing availability, either because the data is copied to different sites allows distribution of queries, either because a site can take over when the main server collapses.

The parallel treatment of access requests improves the performances by letting the users reach the data in a nearby node to avoid the access to the data in a remote location. Replication also provides better fault tolerance [10] and better response times.

Creating replicas of an entity is to replicate the structure and condition of replicated entities [5]. Replication allows mainly:

- To increase the availability of data (locally rather than through connections of networks enabling the access to the objects even if there are some nodes that are inactive [6], which gives the greater robustness system [5].
- To offer a better performance by letting users access data at a neighboring node to prevent access to data in a remote site, and therefore guarantees a better load balancing in the network [5,6].
- To improve reliability and operating safety.
- Thus better protection against corruption.

However, with the use of replication comes the issue of consistency. Consistency is a relationship that defines the degree of similarity between the replicas of an object [7]. Mechanisms to manage the consistency, also called coherence protocols, should be present in all replication system which managed objects can be modified. These mechanisms must implement concrete solutions for scheduling requests for access to different instances of the same object and the propagation of changes made to a replica to the other. Several types of consistency (also called consistency models) exist in the literature, which lead to different degrees of divergence replicas.

Insuring data consistency at all times by means of synchronous replication results in very high operation latencies and thus in bad performance [8]. Moreover, cloud storage systems are deployed on a wide area scale and data are replicated over geographically distant areas. Consequently, latencies become even higher when ensuring strong consistency. These high latencies may generate significant financial losses for service providers that use such storage systems. Consequently, cloud providers tend to rely on storage systems with eventual consistency. Eventual consistency allows the system to return some stale data at some points in time, but ensures that all data will eventually become consistent.

While strong consistency guarantees that all replicas of an object are identical from the user's point of view, the low coherence makes some difference. When the replication system ensures a strong coherence, the result of an access request to any one replica reflects the results of all previous amendments to the

application. When the guarantee consistency is weak, an access request does not reflect necessarily all previous changes, but these will be passed after a finite time.

The literature distinguishes two main approaches to manage the consistency of a replicated object. The first, called pessimistic or impatient replication avoids the divergence between the replicas of an object and therefore guarantees a strong consistency. The second, called optimistic or lazy replication allows some divergence between the replicas and therefore ensures weak consistency. Both approaches are used in file systems, databases, and as part of collaborative work [9]. The pessimistic approaches have shown their difficulties to apply to large-scale systems, unlike the optimistic approaches that do not propagate the updates of instantaneously. In order to contribute to the resolution of the consistency of replica management problem in the cloud, we proposed in the context of this article, a hybrid model using the versions of vectors between the replicas for consistency management replicas. We improved the approach by the integration of a convergence mechanism and conflict resolution between the replicas that suits the Clouds Computing. This mechanism is based on the different forms of negotiation between the Local Consistency Service (LCS).

The rest of this paper is organized as follows. Section 2, we summarize important related works. Section 3 presents our hybrid consistency management approach for cloud computing, Section 4 shows experiments and Section 5 concludes the paper.

2. Literature review

Consistency management and its impact on different storage system features, such as performance and availability, was widely studied. We first review the consistency models in distributed systems. Then, we review some research in Grid environments. Finally we review the consistency in a cloud.

2.1. Consistency management in distributed systems

The literature distinguishes two main approaches to managing the consistency of a replicated object [11]. The first, called pessimistic or impatient replication, avoids the divergence between the replicas of an object and therefore guarantees a strong consistency. The second, called optimistic or lazy replication allows some difference between replicas and therefore ensures weak coherence. Both approaches are used in file systems, databases and in the context of collaborative work [12].

Various protocols exist to avoid the divergence of the replicas of an object. They are generally based on locking replicas that are not up to date and release locks as they are. The strong consistency is ensured and users do not observe incoherence in the replicated objects [13].

Several coherence protocols have been proposed that implement the pessimistic replication. Thus, the primary copy protocol elects a replica as the primary copy and the others are secondary. Requests for read access can be performed on any replica while the updates must be performed on the primary copy. When writing, all replicas are locked and the update has propagated from the primary to the secondary copy. If the primary copy becomes faulty, the remaining replicas ensure the election of a new. Simple to implement and manage, nevertheless the primary copy protocol guarantees low availability in writing (limited to a single replica). Thus, other protocols have been proposed that allow updates on multiple replicas while allowing readings on any replica. For example, the Read One Write All protocol (ROWA) [13] allows writes on any reply, the propagation of these to the other replicas is atomic. This atomicity of updates makes it unsuitable protocol when replicas are inaccessible.

The extension proposed by the Read One Write All Available protocol (ROWAA) tries to solve this problem by ignoring the replicas that do not respond during the propagation of an update. It is then necessary to update these replicas with the current state when they become available. This protocol tolerates $n - 1$ failures, where n is the number of replicates, when these do not partition the network. In case of partition, no recovery mechanism is provided. In addition, quorum based protocols have been proposed. A quorum is a minimum set of nodes which have to give their agreement for an access request to an object to be served. The size of the quorum is generally different for reading and writing requests and can also change from one object to another. The quorums are calculated so as to assure that a request for access always has for result the most current replica. Compared to ROWAA, they decrease the number of messages to be exchanged during a writing (less than n messages, where n is the number of replicas) and some tolerate network partitions by reconfiguring quorums available in case of disconnection.

The optimistic approach consciously accepts concurrent access in reading and writing on different replicas of the same object [5]. When requesting access, it is possible to read or write a replica that is not up to date and mechanisms are generally set up in order to reconcile divergent replicas and to solve the detected conflicts during the reconciliation. The optimistic replication has been proposed in many areas such as distributed file systems, databases, or as part of collaborative work [15,16].

In the area of file systems, Coda has proposed the use of optimistic replication [17]. It uses vectors of version to detect conflict between replicas of a file. To allow the automatic conflict resolution, users can associate to files a conflict resolution software (Application Specific Resolvers (ASRs). Upon detection of a conflict, Coda locates the associated ASR file and executes it.

Bayou [18] is one of the most outstanding works in the field of databases. It was designed to databases with mobile users. The reconciliation process is based on the transfer of operations performed on the replicas. Each node executes access requests in an arbitrary order; the transactions are considered provisional. The global serialization order is the order of execution on a site designated as primary. The others cancel their provisional states and re-execute the committed transactions in the order of their validation on the primary site.

Dynamo [22] is a storage service used at Amazon. It is a key-value store used for low latency access to data. It has a peer-to-peer (P2P) architecture that uses consistent hashing to spread the load and a gossip protocol to guarantee eventual consistency [23]. Amazon's shopping cart and other services (e.g. S3) are built on top of Dynamo.

PNUTS [24] is a similar storage service developed by Yahoo! outside the Hadoop project. PNUTS allows key-value based lookup where values may be structured as typed columns or "blobs". PNUTS users may control the consistency of the data they read using "hints" to require the latest version or a potentially stale one. Data is split into range or hash tablets by a tablet controller. This layout is soft-cached in message routers that are used for queries. The primary use of PNUTS is in web applications like Flickr.

PNUTS uses a "per-record timeline" consistency model which guarantees that every replica will apply operations to a single record in the same order. This is stronger than eventual consistency and relies on a guaranteed, totally-ordered-delivery message brokering service. Each record is geographically replicated around several data centers. The message broker does not guarantee in order delivery between different data centers. Thus each record has a single master copy which is updated by the broker, committed and then published to replicas. This makes sure that every update gets replayed in a canonical order at each replica. In order to reduce latency the mastering of the replica is switched according to changes in access location.

RedBlue consistency [25] was introduced in order to provide as fast responses as possible and consistency when necessary. It provides two types of operations: Red and Blue. Blue operations are executed locally and replicated lazily. Therefore, their ordering can vary from site to site. In contrast, Red operations require a stronger consistency. They must satisfy serializable ordering with each other and as a result generate communication across sites for coordination. However, with the RedBlue consistency, Blue operations might have different orders in different sites. Therefore, non-commutative operations executed in a different order won't allow replicas convergence.

RedBlue consistency provides the so needed adaptivity for systems that require performance while do not require a strictly strong consistency. However, one difficulty that arises is the categorization of operations in Red and Blue. It is difficult to intuitively define for each operation its color without a huge analysis effort. Furthermore, and considering that target applications are large–scale applications and geographically distributed, it is fair to assume a high probability for them to operate on large volumes of data and various data types. This in turn, makes the categorization of operations on data extremely difficult showing a big necessity for automation.

2.2. *Consistency management in the Grid*

Consistency has been an active research area in Grid environments.

In [14], they describe model double-layered adapted to the applications on a large scale and which represents the support of the hybrid approach of consistency management of replicas based on pessimistic and optimistic approaches. This hybrid approach present an adapted mechanism based on the various negotiation forms between virtual consistency agents to be able to reduce the number of conflicts between replicas in data grids. To resolve conflicts, they use only the update operations based on the replica version; the replica which has the largest version is selected and its content is sent to other; however, the use of rollback operations is essential to resolve certain conflicts.

Reference [19] introduces the Replica Consistency Service (RCS), which was developed by the European Data Grid Project. The responsibility of RCS is to maintain consistency among existing replicas. To guarantee local consistency, the Local Consistency Service is performed within each storage unit. RCS forces on the consistency problem considering replica metadata. A file locking mechanism is also proposed in RCS for serialization of file access. However, they have not provided any details of the detection process of a divergence or conflict between two replicas. Both local consistency service and consistency service were briefly explained and there is no information on how the use of these services can ensure consistent replicas.

In [20] paper, they formulate the data replication problem and design a distributed data replication algorithm with consistency guarantee for data grid. The approach consists of systematically organizing the data grid sites into distinct regions. the main contributions are: (i) they proposed a new replica placement policy, which determines how many replicas to create and where to place the replicas; (ii) they proposed a new quorum-based data replication protocol; The quorum serves as basic tools for providing a uniform and reliable way to achieve consistency among the replicas of the system. (iii) they investigate various tradeoffs in terms of cost, availability and algorithm complexity of the proposed replication scheme. The drawback of the proposed approach is the use of the quorum mechanism for guaranteeing consistency, which makes the system less scalable.

Int [26] compares pessimistic consistency with optimistic consistency, and combines these two existing approaches. It divides replicas into several sites. Optimistic principals are used to ensure replica consistency within each site. Whereas, global consistency is covered by the application of algorithms

inspired from the pessimistic approach. The use of an optimistic consistency in local site, can lead to conflict between representative replicas of sites; however, they manage the conflicts resolution on intra-site only and do not support the conflicts resolution on enter-sites;

2.3. Consistency management in the cloud

As Cloud Computing technology emerges, more and more cloud storage systems have been developed. Systems that implement strong consistency serve many applications including services such as mail service, advertisement, image hosting platforms, data analytics applications, and few cloud services as well. These applications, in general, require strong consistency.

In [27], the authors present MetaStorage, a federated Cloud storage system that can integrate diverse Cloud storage providers. According to the authors, MetaStorage is a highly available and scalable distributed hashtable that replicates data on top of diverse storage services. MetaStorage reuses mechanisms from Amazon's Dynamo for cross-provider replication and introduces provider configurations as a new means beyond existing configurations of traditional quorum systems and thus provides additional control mechanisms to manage consistency-latency tradeoffs. There are some problems which MetaStorage cannot fix: it introduces drawbacks compared to using a single Cloud storage service. MetaStorage causes monetary overhead due to redundant data storage. It causes also a latency overhead because it adds an additional layer to the system stack that must be traversed for every service invocation. In addition, MetaStorage relaxes consistency guarantees by Dynamo-style replication protocols. However, it does not provide any conflict resolution mechanism.

[28] proposed to use the concept of Data Consistency Plan (DCP) to define the consistency requirements for SaaS services, and proposed to use a new platform service (i.e., Data Consistency as a Service (DCaaS)) for executing such DCP plan to decouple SaaS developers from managing data uncertainty issues in their code. We also proposed a quota-based approach for managing data uncertainty on eventually consistent cloud data stores. The proposed approach ensures global data consistency by distributing the capacity of strong consistency data objects among datacenters, and then adopts a lazy replication approach for synchronizing the data stores. The disadvantage of the approach that the consistency level is selected manually; Service developers will define the consistency requirements for each data objects.

[29] they introduce new formalism for describing services in service-oriented replication. They propose the notion of consistency regions and relevant service oriented requirements policies, by which trading between consistency and scalability requirements can be handled within regions. The downside of the approach is that the category of services and the regions they are defined by the system administrator, a wrong classification, affect consistency guarantee.

The strategies, proposed by Wang et al. [30], are based on a file's reading and writing frequencies, so that when a system is running, an appropriate consistency strategy is adopted according to the dynamic running status of system and a balance is struck between overhead and performance. However, adjusting parameters is not a simple task, the proposed mechanism allows system to automatically switch consistency strategies according to only update frequency and read frequency. They do not consider more factors for switching strategy, for example according to the storage system state and the application specified requirements.

[31] propose self-adaptive methods that tune consistency levels at runtime in order to achieve better performance, availability and reduce the monetary cost without violating the consistency requirements of the application. Furthermore, they introduce a behavior modeling method that automatically analyzes the application and learns its consistency requirements. They proposed a novel approach [32], named

Harmony, that automatically tunes the consistency level at runtime according to the application requirements. Harmony monitors the storage system and data accesses in order to estimate the stale reads rate in the system. Accordingly, it scales up/down the consistency level to preserve a stale rate tolerated by the application. The proposed approach does not give any guarantees on when all replicas in the storage system would converge to a consistent state. This could be a real problem for many applications as there is no provided certainty about read data. Moreover, if the read data is stale, the question is how stale it could be. Furthermore, the proposed system might be improved by adding different levels of guarantees considering the network performance and topology in addition to data location.

[33] propose a self-adaptive replica consistency strategy which allows the system to automatically select the adequate consistency strategy according to file heat. Based on the impact of file access time sequence, they propose a file heat calculation method, the file heat calculated by which can accurately predict the future file access behavior. They introduce a new algorithm MRFU (Most Frequently and Recently Used) for the calculating of file heat, by which the file used most recently and frequently in a time period is assigned the largest heat value. The author uses several main replicas and several sub-replicas, they have not addressed the case of modification of several sub-replicas at the same time, and the proposed approach does not support the global consistency. They did not describe how to resolve conflicts in the case of use of eventual consistency strategy.

[34] proposed an automated and Self-Adaptive Approach that tunes the consistency level at runtime to reduce the probability of stale reads caused by the dynamicity of cloud systems and the application demands thus providing adequate tradeoffs between consistency and both performance and availability, they presented Adaptive Consistency Model (ACM), The goal of ACM is to dynamically and elastically handle consistency at run time, in order to provide adequate tradeoffs between consistency and both performance and availability. Accordingly, ACM considers not only the application requirements but also the storage system state. Moreover, rather than relying on a standard model based only on the access pattern to define the consistency requirement of an application. Authors have failed to show how to choose the level of consistency and they did not present any formula to calculate the threshold. In addition the use of multiple consistency levels involves the presence of conflicts between replicas; the authors did not mention how to solve this problem.

In the end, we find in the literature two conflicts detection approaches: syntactic [17] and semantic detection [21]. The syntactic approach uses the execution order of operations and the semantic approach uses the knowledge of the semantics of operations to detect conflicts. Regarding the resolution of conflicts, it can be manual or automatic. The first type has two possible versions of the replica. From the two versions of the object, so a user can create a new or merge versions or cancel effects and resubmit the operation. Automatic conflict resolution is performed by a specific procedure for the application that takes the two versions of an object and proposes a new one. Generally, it is based on constraints or preconditions to the transaction.

3. Proposed approach

The approach for replicas consistency management for large-scale systems that, we propose is based on modeling of a cloud in the form of a hierarchical model with two-level (see Fig. 1).

Each datacenter contains a set of host and each host is composed essentially of two types of resources: Computing Elements (CE's) and Storage Elements (SE's). Replicated data are stored on SE's and accessed from CE's via reading or writing operations. Each replica attached to additional information is

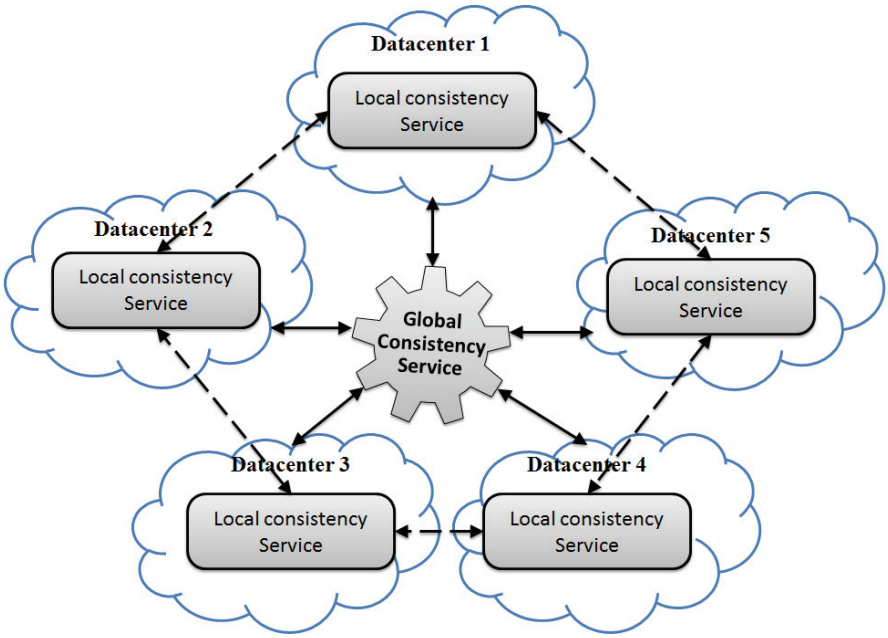


Fig. 1. Hierarchical model for replicas consistency management.

called metadata. The latter gives a high description of replica and it can contain several additional information on the state of this replica, for example, number of versions, timestamp, origin of the update, factor of priority, etc.

We define k Local Consistency Service (LCS's) each one corresponding to each datacenter of a Cloud. A Local Consistency Service *LCSi* is responsible:

- To manage replica consistency within a datacenter *DCi*.
- To cooperate with others *LCSj* to ensure a replica consistency for the whole cloud.

3.1. Local consistency manager

The local consistency manager is the group of cooperating agents coordinated by three agents, the *Monitor*, Consistency and Conflict Management agents.

3.1.1. Monitor agent

The monitor agent collects relevant metrics needed to manage the local consistency. It collect the number of reads and writes in storage element. The monitor agent was designed in a multithreaded manner in order to make it time efficient and to reduce the monitor time. Each thread collects data from a set of hosts and at the end an aggregation process is applied. The monitor time is measured and taken into account when computing the read rates and write rates. This data is further communicated to the adaptive consistency agent. This agent is also responsible for monitoring the charge of the datacenter (Fig. 2).

3.1.2. Consistency agent

We have defined three levels of consistency for datacenter:

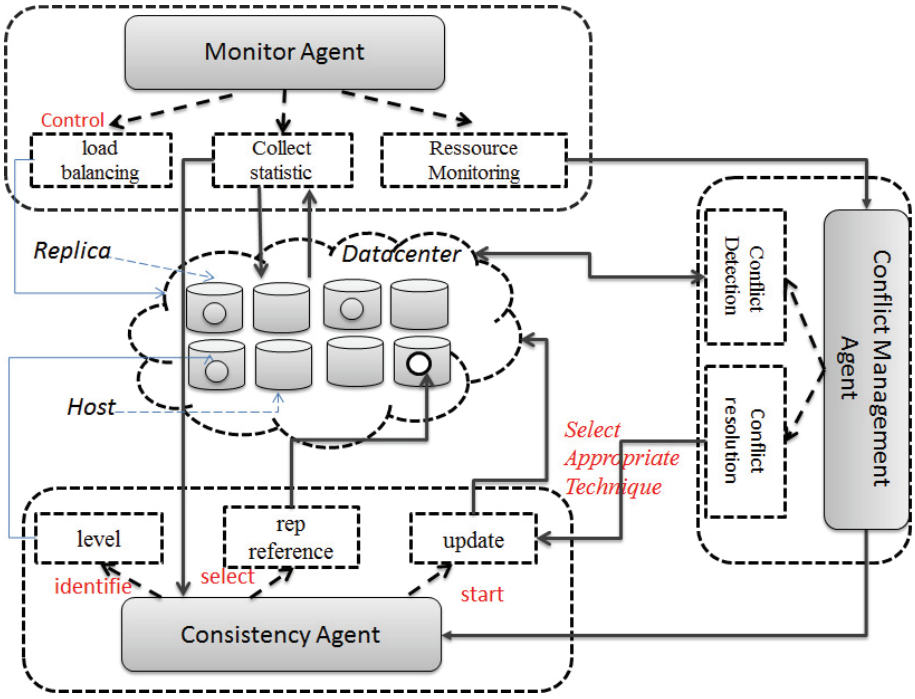


Fig. 2. Local consistency manager architecture.

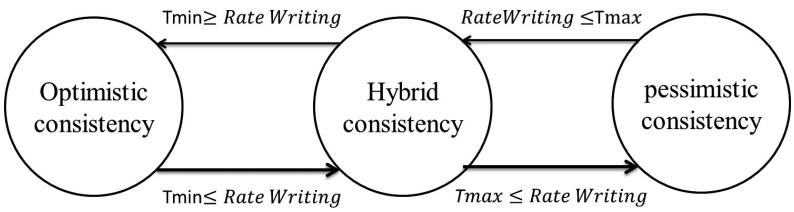


Fig. 3. Datacenter consistency levels.

- Optimistic consistency: the write rate is lower than the threshold *Tmin*.
- Hybrid consistency: the write rate between *Tmin* and *Tmax*.
- Pessimistic consistency: when the write rate exceeds *Tmax*.

Two system parameters are used: the write thresholds *Tmin* and *Tmax*, these parameters determine the margin of all three levels as shown in Fig. 3.

The main functions of consistency agent are: working with different consistency managers to converge the replicas of the same file to a global reference replica; they disseminate new information of the reference replica at different nodes within its site.

These two agents are complementary and the main objective to converge all replicas to a global reference replica.

3.1.3. Conflict management agent

The conflict management agent is responsible cooperation and negotiation while each agent performs its assigned subtask. The main functions of conflict management agent are:

a) Conflict detection:

Each conflict management Agent automatically detects and resolves replication conflicts when they occur. For example, when a master site pushes its deferred transaction queue to another master site in the system, the remote procedures being called at the receiving site can automatically detect if any replication conflicts exist.

Conflicts are detected between two sites i and j as follows:

(i) If *VVi* = *VVj*, then the replicas have not been modified.
(ii) Otherwise, if *VVi* dominates *VVj*, then i is newer than j; that is, Site i has applied all the updates that Site j has, and more. Site j copies the contents and VV from i. Symmetrically, if *VVj* dominates *VVi*, the contents and VV are copied from j to i.
(iii) Otherwise, the operations are concurrent, and the system marks them to be in conflict.

b) Critical situations:

The algorithm to check the existence or not of critical situation of *LCSi*, is given in Algorithm 1. If *LCS* is in critical situation, i.e., that one of these events is detected, then it starts the process of negotiation, see Algorithm 2. To study the evolution of the divergence of replicas inside a datacenter, we put forward the following measures:

Algorithm 1: Local critical situation

```
Let τ^m, D^m_l be parameters of tolerance
/* τ^m: Rate of conflicts number tolerated */
/* D^m_l: Local distance tolerated */
Begin
   Calculate: τ_i, D_local for LCSi
   If (τ_i > τ^m) or (D_local > D^m_l) Then
         return true //existence of critical situation of LCSi
   else
         return false
   end if
end
```

Measure rate of the number of conflicts per datacenter (τ_i): this measurement makes it possible using LCS to know the rate of conflicts by the total number of replicas of the same data inside a datacenter, and it is given by:

$$\tau_i = \frac{\textit{Conflicts\_Nbr}\,(\textit{LCS}_i)}{n_i} \tag{1}$$

Where $\textit{LCS}_i$ is the identifier of i-th *LCS* and n_i is the number of replicas in datacenter LCS_i of given object.

Measure distance within a datacenter (*Dlocal)*: we define Dlocal measurement between the versions maximum and minimal of replicas of the same object inside a datacenter of a *LCS*. This measurement makes it possible to give a vision on the age of the replicas. It can be to calculate by:

$$D_{\textit{local}}\,(\textit{LCS}_i) = \textit{Max}_{t=1}^{n_i}\,(V_{it}) - \textit{Min}_{t=1}^{n_i}\,(V_{it}) \tag{2}$$

We detect critical situations of one *LCS* to the meeting of one of the following cases:

$$\tau i > \textit{Rate of conflicts number tolerated} \tag{3}$$

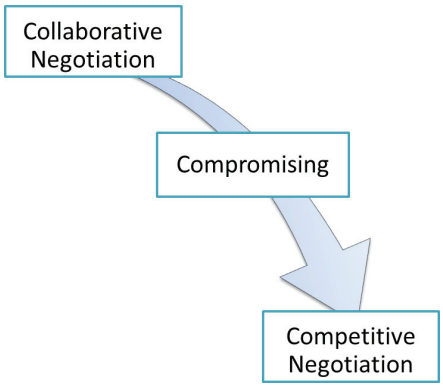


Fig. 4. Conflict resolution.

or

$$Dlocal(LCSi) > Distance\ tolerated \tag{4}$$

c) Conflict resolution:

After a conflict has been detected, resolve the conflict with the goal of data convergence across all sites. In order to resolve a conflict situation, a negotiation process among all conflict parties has to take place. The result of the negotiation should be a solution that results in goal achievement for all involved agents. To use a conflict resolution mechanism in different execution environments the proposed mechanism consists of several negotiating strategies (Fig. 4).

We will describe, in what follows, the core of the consistency process based on the negotiation between LCS (see Algorithms 2–4). We will start by giving definitions of some basic elements, then we will present the mechanism of negotiation used for the conflicts resolution between the replicas.

Firstly, we choose replicas that have made most updates operations compared to other replicas and add them to the set of candidates' replica (Algorithm 2).

Algorithm 2: Search the candidates' replica

Let R a set of replicas of the same object

$R = \{r_1, r_2, \ldots, r_n\} \quad n \geqslant 2$

For all replica $r_i \in R$

Calculate the average of version vector VV_i//VV_i represent version vector of the replica r_i

$$averVV_i = \frac{\sum_{k=1}^{n} VV_{r_i}[k]}{n}$$

End For

Search the candidates' replica rc having maximum average of version vector VV_i among all replicas $r_i \in R$.

$RC = \{Max_{i=1}^{n}(averVV_i)\}$

Let RC a set of candidates' replica

$RC = \{rc_1, rc_2, \ldots, rc_m\} \quad 1 \leqslant m \leqslant n$

Secondly, and in order to search the candidates' replica reference (Algorithm 3), we calculate the number of missing update operations for the r_j replica relative to the candidate replica rc_i; then we calculate the number of update operations that have made in the replica r_j and are missing in the candidate replica rc_i.

Algorithm 3: Search the candidates' replica reference

For all candidate replica $rc_i \in RC$

For all replica $r_i \in R$

$$d\_up\,(rc_i, r_j)_k = \begin{cases} 1, & VV_{rc_i}[k] > VV_{r_j}[k] \\ 0, & else \end{cases} \quad 0 \leqslant k \leqslant n$$

$$D\_Update\,(rc_i, r_j)_{ij} = \sum_{k=0}^{n} d\_up\,(rc_i, r_j)_k$$

$$d\_rollback\,(rc_i, r_j)_k = \begin{cases} 1, & VV_{rc_i}[k] < VV_{r_j}[k] \\ 0, & else \end{cases} \quad 0 \leqslant k \leqslant n$$

$$D\_Rollback\,(rc_i, r_j)_{ij} = \sum_{k=0}^{n} d\_rollback\,(rc_i, r_j)_k$$

End for

End for

Finally, to select the reference replica (Algorithm 4), we calculate the number of update and rollback operation for candidate replica rc_i relative to replica r_j in R; then we search the candidates' replica reference rrc having minimum number of update and rollback operations. The last step is to choose one reference replica using the adequate policies.

Algorithm 4: Select the reference replica

For all candidate replica $rc_i \in RC$

$$D\_Update\,(rc_i) = \sum_{j=1}^{n} D\_Update\,(rc_i, r_j)_{ij}$$

$$D\_Rollback\,(rc_i) = \sum_{j=1}^{n} D\_Rollback\,(rc_i, r_j)_{ij}$$

$$MinD_{Update,Rollback} = Min_{j=1}^{m}\,(D\_Update\,(rc_j) + D\_Rollback\,(rc_j))$$

$$RRC = \bigcup_{i=1}^{m} \{rc_i / (D\_Update\,(rc_i) + D\_Rollback\,(rc_i)) = MinD_{Update,Rollback}\}$$

If $card\,(RRC) = 1$ **than**

$\exists$ *One candidate reference replica* $rrc_i \in RRC$

Else if $card\,(RRC) > 1$ **than**

Let RRC' a set that contain rrc with the minimum of rollback

If $card\,(RRC') = 1$ **than**

$\exists$ *One candidate reference replica* $rrc_i \in RRC'$

Else if $card\,(RRC) > 1$ **than**

We use a politic to select the reference replica for example priority/ID/last write

End if

End if

3.1.4. Negotiating strategies

Now, we will describe the various negotiating strategies:

(i) Collaborating/collaborative negotiation

Collaboration works by integrating all updates operations that have been made in multiple replicas. The object is to find a creative solution acceptable to everyone. Collaboration, though useful, calls for a significant time commitment not appropriate to all conflicts.

Example 1. Collaborative negotiation

R1	R2	R3	R4
1	0	1	0
1	1	1	0
1	1	0	1
0	0	1	1

D(R2,R1) = 1 update
D(R3,R1) = 1 update, 1 roll back
D(R4,R1) = 2 updates, 1 roll back

D(R1,R3) = 1 update, 1 roll back
D(R2,R3) = 2 updates, 1 roll back
D(R4,R3) = 2 updates, 1 roll back
R3 and R1 considered as a *candidate reference replica*
The Reference Replica: R1

In Example #1, we assume that we have four replicas of the same object; the left part of the example shows the vector version of each replica after several modifications, when the agent detects a conflict, it will try to solve this conflict so it will try to use the collaborative negotiation. Initially Algorithm 2 is used to search the candidates' replica. In our example the replica R1 and R3 will be considered as candidates' replicas since they received the most modification; then we search the candidates' replicas reference using Algorithm 3. Finally, we will apply Algorithm 4 to select the reference replica, in this case the replica R1 is chosen as reference replica; the selected reference replica can needs to roll back to minimize the total number of updates and then the state of this replica will be sent to all other consistency agent to update their local replicas.

(ii) Compromising

The compromising strategy typically calls for both sides of a conflict to give up elements of their position in order to establish an acceptable, if not agreeable, solution. This strategy prevails most often in conflicts where the parties hold approximately equivalent power (priorities).

(iii) Competing/competitive negotiation

Competition operates as a zero-sum game, in which one side wins and other loses. Highly assertive personalities often fall back on competition as a conflict management strategy. The competitive strategy works best in a limited number of conflicts.

Invalidate Both: Following this inconsistency resolution policy, both of the conflicting versions are considered invalid and they will be rolled back to a previous consistent state. That state is thus defined as the reference consistent state.

ID-Based: In this policy, each node is assigned a random ID. When detecting a conflict of version vectors, the system chooses the node with a larger ID and considers its replica to be in the reference consistent state.

Priority-Based: This policy assigns different priorities to users. When a conflict arises, the higher-priority user wins and his or her replica is chosen as the reference.

Example 2. Competitive negotiation

R1	R2	R3	R4
1	0	1	0
1	2	1	1
1	1	2	1
0	1	0	1

D(R1,R2)= 1 update
D(R3,R2)= 2 updates, 2 roll back
D(R4,R2)= 2 updates, 1 roll back

D(R1,R3)= 2 updates, 1 roll back
D(R2,R3)= 2 updates, 2 roll back
D(R4,R3)= 1 update

R3 and R2 considered as a *candidates reference replica*

Reference replica is selected according to: Priority / ID / last write

In Example #2, we will also take 4 replicas of the same object; after construction of the reference vector using the vector version of replicas, the agent will detect a conflict; to resolve this conflict, we will use Algorithms 2 and 3 to search candidates' replicas reference. In our case, replica R3 and R2 are considered as candidates' replicas reference. We use now Algorithm 4 to select the reference replica; as the two replicas have the same number of update and roll back compared to other replicas. Reference replica is selected according to: Priority/ID/last write and the state of the reference replica is sent as it is without rollback to all other consistency agent.

3.2. Global consistency manager

The objective of the manager is to achieve global consistency, which is to converge the replicas of a file O_k, stored in different datacenter, towards a global reference replica.

The main steps of inter-datacenter consistent processes are defined as follows:

Step 1. Selection of reference of the replicas: Each datacenter *DCi* selects a replica reference for all instances of the same file O_k stored in the datacenter *DCi*;

Step 2. Distribution: Each replicas manager, distributes the vector versions of its replica reference to other replicas managers of level 1 of the proposed consistency management model;

Step 3. Comparison and upgrade: This step can be decomposed as follows:

- Construction of a reference vector *VREF*;
- Detection of convergences, differences and conflicts between replicas managers based on the *VREF* vector;
- Conflict resolution and upgrading replica managers;

Step 4. Local broadcast: This step is to broadcast new information (vector versions, set of updates) of each replicas reference to replicas of the same file O_k stored in a datacenter *DCi*.

4. Simulation and result

The recent efforts to design and develop Cloud technologies focus on defining novel methods, policies and mechanisms for efficiently managing Cloud infrastructures. Access to the infrastructure in Cloud Computing incurs payments in real currency; to test these newly developed methods and policies, researchers need tools that allow them to evaluate the hypothesis prior to a real deployment in an environment, where one can reproduce tests.

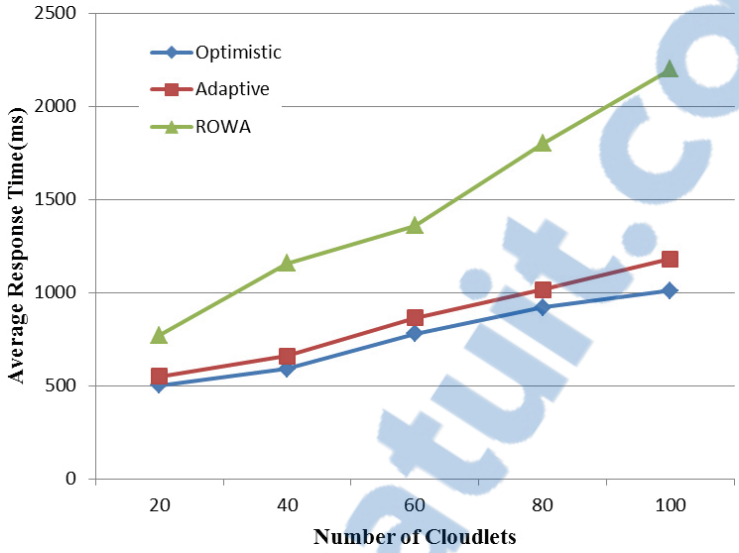


Fig. 5. Variation in average response time compared to the number of cloudlets.

Simulation-based approaches in evaluating Cloud computing systems and application behaviors offer significant benefits, as they allow Cloud developers: (i) to test performance of their provisioning and service delivery policies in a repeatable and controllable environment free of cost; and (ii) to tune the performance bottlenecks before real-world deployment on commercial Clouds. To test our approach we extended CloudSim (v3) to support replica consistency. CloudSim is a generalized and extensible simulation framework that enables seamless modelling, simulation, and experimentation of emerging Cloud computing infrastructures and application services [35].

To analyze the results relating to experimentation of our approach, we used three metrics namely the response time, the number of conflicts between replicas and the number of update propagations. These metrics were used to evaluate the *proposed approach* compared to the optimistic approach and ROWA approach. In all experiment we used a fixed the number of replica and they are placed in the most popular hosts (that receive more requests) to minimize the data access time and network load. In this section we present the results of the various tests that we performed on the CloudSim simulator.

4.1. Experiment 1: Impact of the number of cloudlets

In this first series of experiments, we wanted to measure the impact of the number of Cloudlet on different metrics mentioned in the previous section. These experiments were performed with the following simulation parameters: two datacenters, 20 hosts, 1 data and 20 replicas. Regarding the cloudlets, we have considered a number of cloudlets ranging from 20 to 100 by steps of 20. The results of these experiments are summarized in Figs 5–7.

4.1.1. Effects on average response time

The Response time metric measures the response time relating to the execution of a Cloudlet; We note that in Fig. 5, regardless of the number of executed cloudlets, the average response time of the pessimistic approach: ROWA is clearly more than the Optimistic and Adaptive approaches. This is justified by the fact that this approach blocks writing request, which has the effect of increasing the execution time of a Cloudlet. We also note that the performance of optimistic approach is better than the proposed approach.

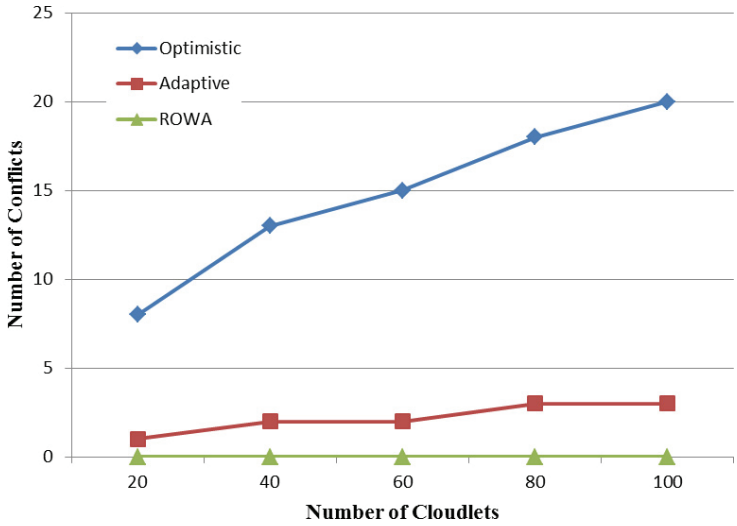


Fig. 6. Variation in number of conflicts compared to the number of cloudlets.

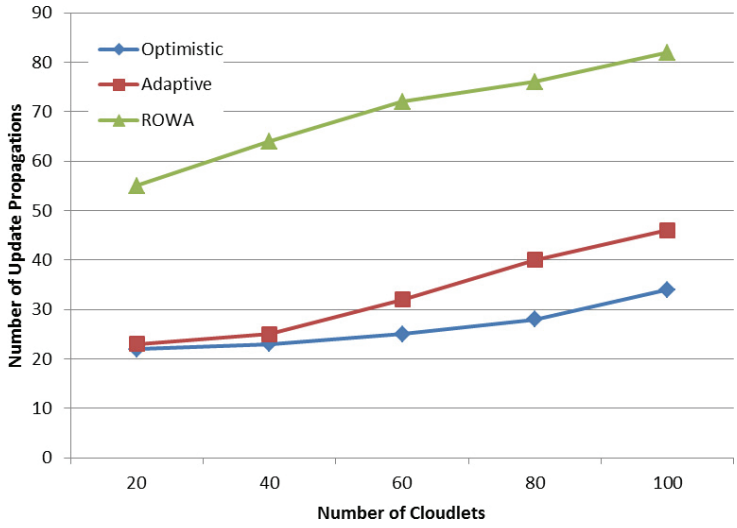


Fig. 7. Variation in number of update propagations compared to the number of cloudlets.

4.1.2. Effects on the number of conflicts

One of the major problems encountered in the propagation of updates phase relates conflict detection. Control and resolve this is essential for the management of consistency. Two replicas of the same file are said to conflict if the difference between their version numbers includes more than two components.

In Fig. 6, we focused on the evolution of the number of conflicts relative to the number of cloudlets. The analysis of results of this figure shows a decrease in the number of conflicts in the proposed approach.

4.1.3. Effect on the number of update propagations

In the next experiment we observe the effect of cloudlets number on the number of update propagations for different approaches. Figure 7 shows the result of the experiment in terms of number of update

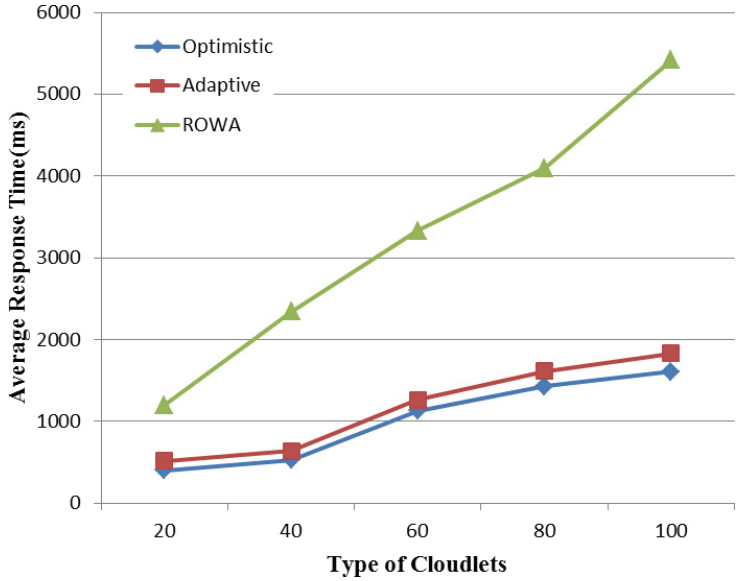


Fig. 8. Variation in average response time compared to the type of cloudlets.

propagation messages in the system as a percentage of all the messages (read, update...). We observe that the number of update propagation messages for ROWA is highest among all the. This is because in ROWA every new writeset is propagated to all the replicas immediately instead of accumulating them. But as the value of the number of cloudlets increases the number of update propagation messages increase. The number of update propagation messages for optimistic approach is better than the adaptive approach when the number of cloudlets is greater than 60 and it remains almost the same for the approach when the number of cloudlets is lower than 60.

4.2. Experiment 2: Impact of type of cloudlets

In this second series of experiments, we wanted to test the impact of the type of cloudlets. For this, we used the following simulation parameters: 2 datacenters, 20 hosts, 1 data and 20 replicas. Regarding the cloudlets, we set the number of Cloudlet 100 and we varied the type of access from 20% writing to 100% writing. The results of these experiments are illustrated in Figs 8–10.

4.2.1. Effects on the average response time

The analysis of the results in Fig. 8 shows that when the number of writes increases relative to the number of reads, the average response time increases, it is because writing is much consumes more in execution time than reading, further pessimistic approach blocks the system to perform the write cloudlets. We also note that the optimistic approach is more efficient than the proposed approach.

4.2.2. Effects on the number of conflicts

The interpretation of the graph in Fig. 9 shows that the adaptive approach could substantially reduce the average number of conflicts over the optimistic approach. We also note that the average number of conflicts increases with the increase of the write rates compared to the reading rate, it is envisaged that the writing Cloudlet leads to conflicts which are due to updates performed on the all of the replicas during the launch of the local and global consistency management process, unlike reading cloudlets who only consult replicas without any modification.

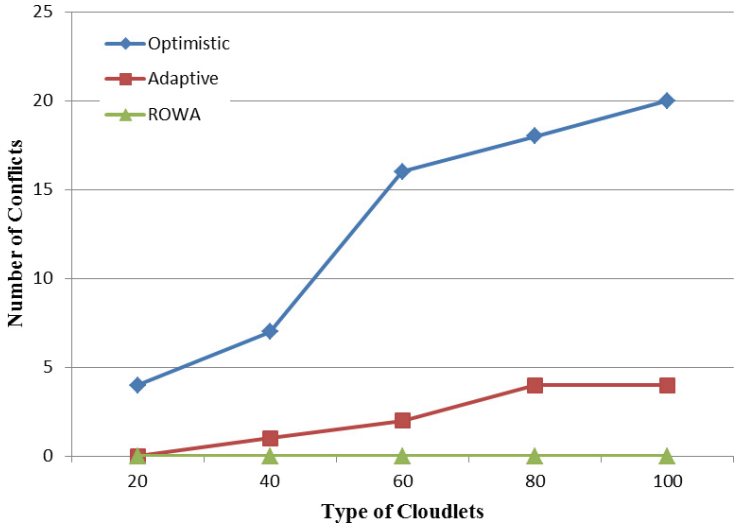


Fig. 9. Variation in number of conflicts compared to the type of cloudlets.

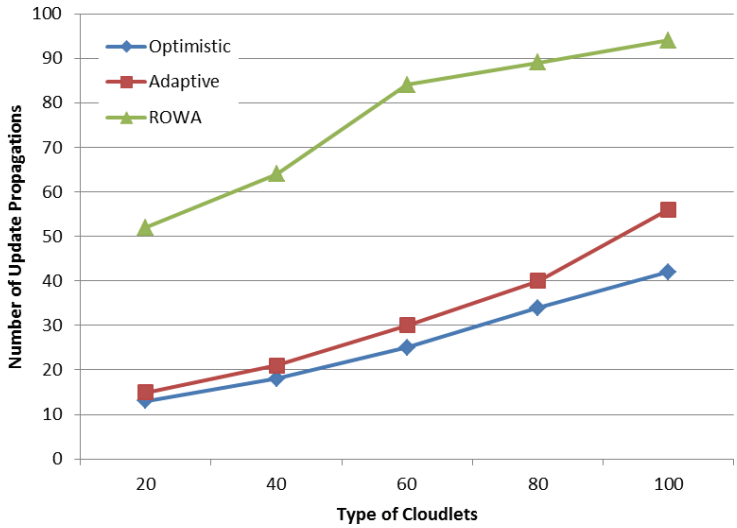


Fig. 10. Variation in number of update propagations compared to the type of cloudlets.

4.2.3. Effect on the number of update propagations

In the last experiment we compare the number of update propagations for different approaches. The experimental setting is the same as the previous experiment, see Fig. 10.

We observe that ROWA approach results in higher number of update propagations than adaptive and optimistic approaches. The explanation is same as the previous Experiment (1.3).

There is no significant difference in the number of update propagations between adaptive and Optimistic approaches when the write rates is lower than 40. This difference increase as the value of the threshold of write increases and the difference is highest when the write rates achieved 100. This because when the threshold of write increases; the adaptive approach makes more update to reduce the number of conflicts.

5. Discussion

Due to the high network latencies, strong consistency guarantees are too expensive when storage systems are geographically distributed. Therefore, weaker consistency semantics such as eventual consistency is the most popular choice for applications that require high availability and performance. However, weaker consistency models are not suitable for all applications classes, even if most operations within one application require only eventual consistency. Many systems implement eventual consistency. These systems are usually destined to serve social networks, web shop applications, document-based applications, and cloud services.

Several studies have been proposed to help to resolve this problem, Most approaches use mechanisms to change the level of consistency during execution. Some works have failed to show how to choose the level of consistency and they did not present any formula or algorithm to select the optimal consistency level. In addition the eventual consistency does not give any guarantees on when all replicas in the storage system would converge to a consistent state. However the use of multiple consistency levels can involves the presence of conflicts between replicas; the majority of work did not describe how to solve this problem; they are content only to detect conflicts.

In our contribution, initially we proposed an approach for ensuring consistency in a flexible manner in the cloud Computing; that tunes the consistency level at runtime. In a second step we have addressed the problem of conflict resolution between replicas; the proposed mechanism consists of several negotiating strategies. Which allows to reduce the number of conflict and the probability of stale reads caused by the dynamicity of cloud systems and the application demands.

Collaboration among agents is the key for our approach; we used it to take advantage of differences and similarities between the replicas. Each replicas manager agent, distributes the vector versions of its replica reference to other replicas managers agents; the replica version vector are used for constructing a reference vector VREF. Which is used by the various consistency manager agents to detect the convergences, differences and conflicts between replicas; in the presence of conflict, the consistency managers agents work with different consistency managers' agents to converge the replicas of the same file to a global reference replica by using the negotiation process; the result of the negotiation should be a solution that results in goal achievement for all involved agents. To achieve collaboration across agents, we used two types of communication among agents, global communication and local communication. Figure 1 shows the global interaction between Local Consistency Service (where each LCS is composed of three agents) and Fig. 2 shows the interaction in the local level.

For applications that require high availability and performance, selecting the number of replicas and their location affect these two criteria; One of the problems of our approach is that we used a fixed number of replicas; and we placed the replicas in the popular nodes beforehand and not running; we have not studied the impact of the variation in the number of replicas on system performance; it would be interesting to study the behavior of our approach in this scenario.

There is another problem, if both parameters Rate of conflicts number tolerated and Local distance tolerated that are used to detect critical situations, are not well defined. Moreover, if the collaborative and compromised negotiation strategies fail to resolve the conflict, the competitive negotiation strategy can generate significant additional cost due to the cancellation of several write. The integration of a module that allows to manage replicas dynamically and the proposal of an algorithm to define the two parameters considering to data type and application requirement can increase the effectiveness of the proposed approach.

6. Conclusion

Intensive data processing applications handle extremely large volumes of data. Moreover, they require very fast treatment times. Much of these applications are deployed on cloud, to take advantage of this infrastructure such as scalability, reliability and high availability at a reduced cost. In this context, replication is an essential way in the cloud. It provides the necessary means to ensure the availability of data across multiple copies, fast access provided via local copies of data, data durability, fault tolerance and disaster recovery by replicating data on several sites.

However, replication introduces the important problem of data consistency. The consistency management is paramount. It has major consequences on distributed storage systems. Thus, and with the large scales of Big Data, manage consistency efficiently is crucial to meet the performance requirements, availability, and costs in the cloud. In the context of this paper, we address at this particular context. We proposed a consistency management solution that is flexible and adaptive to meet the needs of applications.

Initially we proposed an adaptive and hybrid model using the versions of vectors among the replicas for consistency management replicas. In a second step we improved the approach proposed by the integration of a convergence mechanism and conflict resolution between the replicas that suits the Clouds Computing.

To demonstrate the proposed approach we extended the CloudSim simulator and we have launched a series of experiments by creating several scenarios. Performance is evaluated using a set of metrics such as the execution time, the number of conflict and the number of update propagations.

The results obtained by the simulation show the effectiveness of our approach in term of execution time and masking divergence and incompatibility situations.

References

[1] S. Zhang, X. Chen, S. Zhang and X. Huo, The comparison between cloud computing and grid computing, *Proc International Conference on Computer Application and System Modeling (ICCASM)*, Taiyuan **11** (2010), 72–75.

[2] J. Peng, X. Zhang, Z. Lei, B. Zhang, W. Zhang and Q. Li, Comparison of several cloud computing platforms, *Proceedings of the 2009 Second International Symposium on Information Science and Engineering*, Washington, DC, USA, (2009), 23–27.

[3] C. Gong, J. Liu, Q. Zhang, H. Chen and Z. Gong, The characteristic of cloud computing, *IEEE 39th International Conference on Parallel Processing Workshops*, San Diego, CA, (2010), 275–279.

[4] H.-E. Chihoub, Managing consistency for big data applications: Tradeoffs and self-adaptiveness, Ph.D. Dissertation, École normale supérieure de Cachan-Antenne de Bretagne, 2013.

[5] G. Belalem, Z. Benotmane and K. Benhallou, Self adjustable negotiation mechanism for convergence and conflict resolution of replicas in data grids, *International Journal of Cognitive Informatics and Natural Intelligence (IJCINI)* **3**(1) (2009), 95–110.

[6] G. Pucciani, The replica consistency problem in data grids, Ph.D. Dissertation, University of Pisa, 2008.

[7] B.B. Nandi, H.S. Paul, A. Banerjee and S.C. Ghosh, Fault tolerance as a service, *IEEE Sixth International Conference on Cloud Computing*, Santa Clara, CA, (2013), 446–453.

[8] H.-E. Chihoub, S. Ibrahim, G. Antoniu and M. Pérez, Harmony: Towards automated self-adaptive consistency in cloud storage, *IEEE International Conference on Cluster Computing*, Beijing, China, (Sep 2012).

[9] M. Zouari and I.B. Rodriguez, Towards automated deployment of distributed adaptation systems, *7th European Conference on Software Architecture (ECSA)*, Montpellier, France, (1–5 July 2013), 336–339.

[10] S. Limam and G. Belalem, A migration approach for fault tolerance in cloud computing, *International Journal of Grid and High Performance Computing (IJGHPC)* **6**(2) (2014), 24–37.

[11] G. Belalem and Y. Slimani, A hybrid approach for consistency management in large scale systems, *International Conference on Networking and Services (ICNS'06)*, Silicon Valey, CA, USA, (16–19 July 2006), 71–71.

[12] L.J. Kawell, S. Beckhardt, T. Halvorsen, R. Ozzie and I. Greif, Replicated document management in a group communi-

cation system, in: *Second Conference on Computer-Supported Cooperative Work*, Portland, Oregon, (1988), 226–235.
[13] P.A. Bernstein, V. Hadzilacos and N. Goodman, Concurrency control and recovery in database systems, Addison-Wesley, 1987.
[14] G. Belalem and B. Yagoubi, Negotiation mechanisms for resolution conflicts among replicas in data grid, *International Journal of Hybrid Information Technology* **1**(3) (July 2008), 47–60.
[15] P. Reiher, J. Heidemann, D. Ratner, G. Skinner and G. Popek, Resolving file conflicts in the Ficus file system, in: *Proceedings of the USENIX Summer 1994 Technical Conference*, Boston, Massachusetts, USA, (June 1994), 183–195.
[16] P. Kumar and M. Satyanarayanan, Flexible and safe resolution of file conflicts, *Proceedings of the USENIX Winter 1995 Technical Conference on UNIX and Advanced Computing Systems*, New Orleans, Louisiana, USA, (1995), 95–106.
[17] J.J. Kistler and M. Satyanarayanan, Disconnected operation in the coda file system, *ACM Transactions on Computer Systems* **10**(1) (February 1992), 3–25.
[18] K. Petersen, M. Spreitzer, D.B. Terry, M. Theimer and A.J. Demers, Flexible update propagation for weakly consistent replication, in: *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*, Saint-Malo, France, (1997), 288–301.
[19] A. Domenici, F. Donno, K. Paschen, G. Pucciani, H. Stockinger and K. Stockinger, Replica consistency in a data grid, in: *IX International Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT03)*, Tsukuba, Japan, (December 2003).
[20] J.H. Abawajy and M.M. Deris, Data replication approach with consistency guarantee for data grid, *IEEE Transactions on Computers* **63**(12) (December 2014), 2975–2987.
[21] A.-M. Kermarrec, A. Rowstron, M. Shapiro and P. Druschel, The IceCube approach to the reconciliation of divergent replicas, *Proceedings of the Twentieth Annual ACM Symposium on Principles of Distributed Computing*, Newport, Rhode Island, USA, (August 2001), 210–218.
[22] G. DeCandia et al., Dynamo: Amazon's highly available key-value store, in: *Proceedings of Twenty-First ACM SIGOPS Symposium on Operating Systems Principles, SOSP '07*, Stevenson, Washington, USA, (2007), 205–220.
[23] W. Vogels, Eventually consistent, *Queue* **6**(6) (December 2008), 14–19.
[24] B.F. Cooper et al., PNUTS: Yahoo!'s hosted data serving platform, in: *Proc VLDB Endow* **1** (2008), 1277–1288.
[25] C. Li, D. Porto, A. Clement, J. Gehrke, N. Preguiça and R. Rodrigues, Making geo-replicated systems fast as possible, consistent when necessary, in: *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation, OSDI'12*, Hollywood, CA, USA: USENIX Association, (2012), 265–278.
[26] G. Belalem and Y. Slimani, A hybrid approach for consistency management in large scale systems, *Proceedings of the International conference on Networking and Services*, IEEE Computer Society, (2006), 71.
[27] D. Bermbach, M. Klems, S. Tai and M. Menzel, MetaStorage: A federated cloud storage system to manage consistency-latency tradeoffs, *IEEE International Conference on Cloud Computing (CLOUD)*, Washington, DC, (4–9 July 2011), 452–459.
[28] I. Elgedawy, DCaaS: Data consistency as a service for managing data uncertainty on the clouds, *International Journal of Advanced Computer Science and Applications* **4**(5) (2013), 59–68.
[29] T. Chen, R. Bahsoon and A.-R.H. Tawil, Scalable service-oriented replication with flexible consistency guarantee in the cloud, *Information Sciences: An International Journal* **264** (April 2014), 349–370.
[30] X. Wang, S. Yang, S. Wang, X. Niu and J. Xu, An application-based adaptive replica consistency for cloud storage, *Proceedings of the Ninth International Conference on Grid and Cloud Computing* (1–5 November 2010), 13–17.
[31] H.-E. Chihoub, Self-adaptive cost-efficient consistency management in the cloud, *2013 IEEE 27th International Symposium on Parallel &* Distributed Processing Workshops and PhD Forum (IPDPSW), (20–24 May 2013), 2290–2293.
[32] H.-E. Chihoub, S. Ibrahim, G. Antoniu and M.S. Pérez-Hernandez, Harmony: Towards automated self-adaptive consistency in cloud storage, in: *2012 IEEE International Conference on Cluster Computing (CLUSTER'12)*, Beijing, China, (2012), 293–301.
[33] Z. Zhou, S. Chen, T. Ren and T. Wu, File heat-based self-adaptive replica consistency strategy for cloud storage, *Journal of Computers* **9**(8) (August 2014), 1928–1933.
[34] J. Kim and S.-T. Hwang, Adaptive consistency approaches for cloud computing platform, *International Journal of Software Engineering and Its Applications* **9**(7) (2015), 127–134.
[35] G. Belalem and S. Limam, Towards improving the functioning of CloudSim simulator, in: *International Conference on Digital Information Processing and Communications (ICDIPC)*, Ostrava, Czech Republic, (July 2011), 258–267.

Authors' Bios

Said Limam is a PhD candidate in the Department of Computer Science in the Faculty of exact and applied sciences at the University of Oran1 Ahmed Ben Bella, Algeria. He received his M.S. degree in

2012 from the University of Oran. His research interests are: distributed system, grid computing, cloud computing, fault tolerance, data replication and consistency.

Ghalem Belalem Graduated from Department of computer science, Faculty of exact and applied sciences, University of Oran1 Ahmed Ben Bella, Algeria, where he received PhD degree in computer science in 2007. His current research interests are distributed system; grid computing, cloud computing, replication, consistency, fault tolerance, economic models, energy consumption, Big data, IoT, mobile environment, images processing.

Résumé

Le Cloud Computing attire plus d'attention pour sa haute performance et sa disponibilité à faiblecoût. Une façon de fournir la disponibilité et l'évolutivité consiste à utiliser la réplication. Dans un premier temps, nous proposons une stratégie de réplication dynamique qui satisfait simultanément les besoins des locataires en termes de disponibilité et de performances, tout en tenant compte du profitdu fournisseur. La stratégie proposée est basée sur un modèle de coût qui vise à calculer le nombre minimum de répliques requises pour maintenir une disponibilité élevée des données. En outre, la réplication des données et l'ordonnancement des requêtes sont couplées afin de placer ces répliques demanière équilibrée en tenant compte du budget du locataire. D'autre part, l'existence de plusieurs répliques introduit le problème de cohérence de données. Dans un deuxième temps, nous proposons un modèle basé sur une approche optimiste utilisant des vecteurs de versions entre les répliques pourgérer la cohérence des données. Nous améliorons cette stratégie par l'intégration d'un mécanisme deconvergence et de résolution de conflits entre les répliques qui convient au Cloud Computing.

Mots clés :

Cloud Computing; Réplication, avantages économiques; Cohérence des données; Résolution de conflits; Cohérence adaptative; CloudSim; Qualité de service; Performance; SLA.