Contribution à la réplication de données dans les systèmes de gestion de données à grande échelle

Riad Mokadem

MANUSCRIT

En vue de l'obtention de l'

HABILITATION A DIRIGER DES RECHERCHES

Délivré par : *l'Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)*

Présentée et soutenue le *17/11/2020* par :

Riad MOKADEM

Contribution à la réplication de données dans les systèmes de gestion de données à grande échelle

JURY

Esther PACITTI	Professeur des universités, Université de Montpellier	Rapportrice
Mohand-Said HACID	Professeur des universités, Université Claude Bernard, Lyon 1	Rapporteur
Farouk TOUMANI	Professeur des universités, Université Blaise Pascal - Clermont-Ferrand 2	Rapporteur
Claudia RONCANCIO	Professeur des universités, Institut Polytechnique de Grenoble - ENSIMAG	Examinatrice
Franck MORVAN	Professeur des universités, Université Paul Sabatier, Toulouse 3	Examinateur
Jean-Marc PIERSON	Professeur des universités, Université Paul Sabatier, Toulouse 3	Examinateur
Patrick VALDURIEZ	Professeur, Directeur de Recherche, Inria, LIRMM Montpellier	Examinateur
Abdelkader HAMEURLAIN	Professeur des universités, Université Paul Sabatier, Toulouse 3	Parrain de l'HDR

École doctorale et Domaine

MITT - Informatique

Unité de Recherche

IRIT (Institut de Recherche en Informatique de Toulouse) - UMR 5505

Département et Equipe de Recherche

Département de Gestion de Données- Equipe PYRAMIDE

REMERCIEMENTS

Je remercie très sincèrement Esther Pacitti, Professeure à l'Université de Montpellier, Mohand-Said Hacid, Professeur à l'Université Claude Bernard, Lyon 1, et Farouk Toumani, Professeur à l'Université Blaise Pascal, Clermont-Ferrand 2, d'avoir accepté de rapporter sur ces travaux. Mes remerciements vont également aux examinateurs du jury, Claudia Roncancio, Professeure à l'Institut Polytechnique de Grenoble – ENSIMAG, Patrick Valduriez, Professeur, Directeur de Recherche à l'Inria et au LIRMM, Montpellier, Jean-Marc Pierson, Professeur à l'Université Paul Sabatier Toulouse 3 et Franck Morvan, Professeur à l'Université Paul Sabatier Toulouse 3. Merci à tous pour votre expertise, pour l'intérêt que vous portez à mes travaux de recherche et pour l'honneur que vous me faites de participer à ce jury.

Mes remerciements s'adressent à Abdelkader Hameurlain, Professeur à l'Université Paul Sabatier Toulouse 3, responsable de l'équipe PYRAMIDE et garant de mon Habilitation à Diriger des Recherches (HDR). Je tiens ici à vous exprimer ma profonde reconnaissance pour votre soutien. Merci de m'avoir montré le chemin pour y arriver jusqu'à l'HDR. Merci pour tout ce que j'ai appris à vos cotés en ce qui concerne la recherche et la manière de diriger des recherches.

Je tiens évidemment à remercier mes collègues de l'équipe PYRAMIDE, Abdelkader Hameurlain, Franck Morvan et Shaoyi Yin. Merci pour votre soutien, conseils et innombrables échanges. Lors de la rédaction de ce document, vos retours m'ont été d'une aide précieuse. Merci également pour la convivialité de travail de tous les jours.

Mes remerciements s'adressent également à mes collègues de l'IRIT et de l'IMT avec lesquels j'ai des collaborations pour les activités d'enseignement et/ou de recherche. Merci à mes amis/collègues en France et à l'étranger pour leur encouragement. Merci aussi aux doctorants que j'ai encadré et que j'encadre actuellement, avec lesquels j'ai eu le plaisir de travailler.

Enfin, je ne peux terminer ces quelques lignes sans remercier ma famille qui dans l'ombre, m'apporte le soutien et les encouragements nécessaires. Mes pensées vont à ma femme Naima, mes enfants Maelisse et Enzo, mes chers parents Boualem et Fatima ainsi que toute la famille pour leur soutien sans faille.

.

RESUME

Dans ce manuscrit, nous nous intéressons à la conception de stratégies dynamiques de réplication de données, visant l'amélioration des performances dans les systèmes de gestion de données à grande échelle pour des applications décisionnelles.

L'étude de l'état de l'art nous a permis de proposer une classification des stratégies de réplication de données dans les systèmes de grille de données, par exemple en fonction du type d'architecture de la grille, et une autre classification dans les systèmes Cloud, par exemple en fonction de la prise en compte de la profitabilité du fournisseur. Nous avons constaté que seulement quelques stratégies de réplication dans les systèmes Cloud s'intéressent à la satisfaction de l'objectif des performances pour les locataires tout en garantissant un bénéfice économique pour le fournisseur. De plus, la plupart de ces stratégies visent seulement à réduire la consommation des ressources générée par la réplication sans se focaliser sur le coût économique de cette réplication.

Dans cette perspective, nos travaux de recherche visent la conception de stratégies dynamiques de réplication de données prenant en compte le compromis entre la garantie de performances pour les locataires et la rentabilité économique pour le fournisseur. D'abord, la création d'une réplique est envisagée uniquement si l'objectif de temps de réponse n'est pas atteint pour le locataire. Ensuite, une heuristique est proposée afin de trouver un placement acceptable pour cette réplique, qui satisfait cet objectif tout en étant profitable pour le fournisseur. Pour cela, nous avons étendu les modèles de coûts classiques en intégrant les coûts économiques, notamment le coût engendré par la réplication. Enfin, un algorithme d'ajustement dynamique du nombre de répliques permet au fournisseur de s'adapter à la demande des locataires.

Les performances de nos propositions ont été comparées à celles d'autres stratégies de réplication de données proposées dans les systèmes Cloud. L'analyse des résultats a montré que nos propositions ne se contentent pas seulement de satisfaire les objectifs des locataires. Elles prennent aussi en compte le profit du fournisseur, notamment dans des conditions défavorables, par exemple des taux importants d'arrivée de requêtes et des seuils de temps de réponse stricts.

Mots clés. Gestion de données, bases de données, réplication de données, grille de données, Cloud Computing, modèles de coûts, performances, élasticité, profit économique.

TABLE DES MATIERES

LISTE DES FIGURES

LISTE DES TABLES

LISTE DES ALGORITHMES ET HEURISTIQUES

CHAPITRE 1

Introduction

Résumé

Ce manuscrit présente mes travaux de recherche effectués à l'Université Paul Sabatier UPS - Toulouse III. Ces travaux ont été menés au sein de l'équipe PYRAMIDE (*Optimisation dynamique de requêtes à grande échelle*) au laboratoire IRIT (*Institut de Recherche en Informatique de Toulouse*). Actuellement, les travaux de l'équipe se focalisent sur les principaux problèmes d'allocation élastique de ressources pour le traitement et l'optimisation de requêtes faisant référence à de grands volumes de données distribuées à grande échelle. Dans ce manuscrit, je décris uniquement les travaux de recherche portant sur la réplication de données dans les systèmes de gestion de données à grande échelle. Dans ce chapitre, je commence d'abord par décrire le contexte de mes travaux de recherche. Ensuite, je présente les problématiques liées à la réplication de données dans les systèmes de grille de données et dans les systèmes Cloud. Enfin, je présente mes principales contributions de manière synthétique à la fin du chapitre.

Sommaire

1.1 CONTEXTE ET GENERALITES SUR LA REPLICATION DE DONNEES

1.1.1 Contexte

La démocratisation de l'Internet, la popularisation des périphériques (par exemple, les smartphones) et la popularité croissante des services et applications interconnectés (par exemple Internet des objets et les réseaux sociaux) ont conduit à la génération de gros volumes de données. A titre d'exemples, les utilisateurs de YouTube mettent plus de 500 heures de vidéo par minute à disposition d'autres utilisateurs, ce qui nécessite plus d'un péta-octet de stockage supplémentaire chaque jour[1] (une augmentation de 40% en 5 ans) et Facebook rajoute près de sept pétabytes de stockage chaque mois dans ses centres de

[1] https://www.statista.com/ (December 2019)

données[2]. Cette croissance exponentielle du volume de données, notamment avec l'avènement de l'ère du 'Big Data' et de l'intelligence artificielle, contraint inévitablement les capacités de nombreux systèmes traditionnels, tels que les Systèmes de Gestion de Bases de Données (SGBD) centralisés. En effet, la gestion de données volumineuses, hétérogènes et réparties à grande échelle introduit de nouvelles problématiques pour le stockage, l'accès et le traitement de ces données, d'autant plus que les utilisateurs s'attendent à une certaine Qualité de Service (QdS), par exemple en termes de performances et de disponibilité de données. Les systèmes de bases de données parallèles (Dewitt and Gray, 1992; Valduriez, 1993) ont eu un grand succès dans les années 90. Ils ont permis à de nombreuses applications gérant de gros volumes de données de répondre à leurs besoins en termes de hautes performances et de disponibilité des ressources. Néanmoins, ces systèmes sont très coûteux pour les entreprises/utilisateurs et nécessitent d'avoir des compétences de haut niveau permettant leur maintenance et administration.

Dans les années 2000, les systèmes de gestion de données à grande échelle, tels que les systèmes de grille de données (Foster, 2001) et les systèmes Cloud (Buyya et al., 2009), ont permis aux utilisateurs de réduire leurs coûts en termes d'infrastructures. Ces systèmes sont devenus alors incontournable pour les applications nécessitant une importante puissance de calcul et de gigantesques capacités de stockage. Dans de tels systèmes, ces ressources sont mutualisées entre des milliers d'utilisateurs. Néanmoins, une indisponibilité de ces ressources peut mener à une dégradation de la QdS. Des ressources de calcul surchargées, une faible bande passante pour l'accès à des données distantes ou des capacités de stockage limitées constituent des exemples de problèmes menant à cette dégradation. De nouveaux modèles d'allocation de ressources sont alors nécessaires pour l'évaluation des requêtes dans de tels systèmes. Dans ce contexte, la réplication de données, une technique bien connue dans les systèmes distribués classiques, permet de répondre aux problèmes cités ci-dessus.

Dans ce qui suit, nous présentons d'abord la technique de réplication de données telle qu'utilisée dans les systèmes distribués classiques. Ensuite, nous nous intéressons à la réplication de données dans les systèmes de grille de données que nous décrivons dans le chapitre suivant. Nous montrons que les caractéristiques de tels systèmes, par exemple la grande échelle et la dynamicité des nœuds, introduisent de nouvelles problématiques pour la conception de stratégies efficaces de réplication de données. D'un autre coté, les fournisseurs de Cloud proposent des services variés à travers des architectures parallèles. Nous mettons en évidence les caractéristiques fondamentales des systèmes Cloud telles que la gestion élastique de ressources mutualisées entre plusieurs locataires et la tarification de ces ressources suivant le modèle 'pay as you go'. Ces caractéristiques font que les stratégies proposées précédemment dans les systèmes de grille ne sont pas adaptées aux systèmes Cloud. Ensuite, l'étude de l'état de l'art nous permet de proposer de nouvelles classifications de ces stratégies dans les systèmes de grille de données et dans les systèmes Cloud. Enfin, nous mettons en évidence les principes directeurs de la conception de stratégies de réplication de données pour l'évaluation de requêtes relationnelles dans les systèmes Cloud.

1.1.2 La réplication de données: Généralités

La réplication de données consiste à placer (stocker) plusieurs copies d'un objet de données, appelées répliques, sur plusieurs nœuds distincts (Bernstein et al., 1987). Des exemples

[2] https://www.datacenterknowledge.com/ (Juin 2020)

d'objets à répliquer sont les relations et les fichiers. Une granularité de réplication est l'unité de données minimale qu'on réplique dans un système de gestion de données (Kunszt et al., 2005). Dans la suite de ce manuscrit, un fragment de relation (qu'on appellera relation pour simplifier) constitue la granularité de réplication. Par ailleurs, le facteur de réplication, correspond au nombre total de répliques d'un objet dans le système. Il est également utile de préciser que la réplication de données est souvent à l'initiative du serveur et doit être transparente pour les applications. Enfin, nous verrons que lorsque la réplication intervient dans un système déjà opérationnel, c'est-à-dire lors de la réception des requêtes des utilisateurs, on parle alors d'une réplication dynamique.

La réplication de données vise à satisfaire différents objectifs : (i) l'augmentation de la *disponibilité* de données, (ii) l'amélioration des *performances* d'accès en réduisant les coûts de communication (à travers le placement des répliques près des utilisateurs), en exploitant le parallélisme (à travers l'accès parallèle à différentes répliques situées sur différents nœuds), ou encore en améliorant l'équilibrage de charge (à travers la réplication des données fortement accédées sur différents nœuds), (iii) l'augmentation de la *fiabilité* de données en fournissant des répliques appropriées de données si une partie de ces données est corrompue, et (iv) l'amélioration de la *tolérance aux pannes* en cas de défaillance de nœuds.

Lorsque l'optimiseur d'un SGBD fourni un plan d'exécution d'une requête, la QdS attendue par un utilisateur n'est pas forcément satisfaite pour diverses raisons. Dans ce contexte, la réplication de données permet de répondre à cette dégradation de la QdS suivant différents scénarios :

(i) Une déconnexion d'un nœud contenant la seule copie d'une relation peut nuire de manière durable à la disponibilité de données. Répliquer cette relation sur d'autres nœuds permet de garantir une plus grande disponibilité pour cette relation.

(ii) L'accès à une relation distante à travers un réseau à faible bande passante peut conduire à un temps de réponse insatisfaisant. Répliquer cette relation sur un nœud disposant d'une plus grande bande passante avec le nœud qui requiert cette relation permet de réduire ce temps de réponse.

(iii) La surcharge d'un nœud peut conduire à un temps de réponse dégradé. Cela peut être dû à des requêtes complexes ou à un nombre important de requêtes concurrentes, exécutées sur un nœud ne disposant pas d'assez de ressources. Répliquer les relations accédées sur un nœud moins chargé permet, par exemple, de résoudre ce problème.

Une solution naïve consiste à répliquer les données sur tous les nœuds d'un système. Néanmoins, cette solution n'est pas réaliste vu qu'elle génère une consommation importante de ressources en termes de bande passante du réseau et de stockage. Définir des stratégies de réplication constitue alors la solution à apporter à ces problèmes (Ranganathan and Foster, 2001). Une stratégie de réplication de données doit absolument répondre aux problématiques suivantes :

(i) Quand créer/supprimer une réplique ? Répliquer trop tôt ou trop fréquemment entraîne une utilisation inefficace des ressources et réduit les performances. Une réplication des données paresseuse ou trop tardive est également préjudiciable, car cela neutralise les avantages de la réplication.

(ii) Quelles données sont répliquées ? Cela consiste à identifier les données à répliquer suivant l'objectif de la stratégie de réplication.

(iii) Où placer une nouvelle réplique ? Cela permet d'identifier les nœuds potentiels qui peuvent recevoir les nouvelles répliques. Ces nœuds doivent notamment disposer d'un espace de stockage suffisant et ne doivent pas être surchargés. La bande passante entre le nœud de placement et le nœud hébergeant les données à répliquer doit également être acceptable.

(iv) Combien de répliques sont à créer ? Cela consiste à déterminer le nombre de répliques nécessaires pour la satisfaction de la QdS.

Dans la littérature, de nombreuses stratégies de réplication de données ont été proposées pour l'amélioration des performances des requêtes en lecture seule (Edwin et al., 2019; Mokadem and Hameurlain, 2020). D'autres stratégies visent la satisfaction d'autres objectifs, par exemple la disponibilité de données, tout en assurant une cohérence de données lors des requêtes de mise à jour (Hsu et al., 2018). Dans ce dernier cas, les performances peuvent être dégradées si les données sont fréquemment mises à jour. Les avantages de la réplication peuvent alors être neutralisés par la surcharge liée au maintien de la cohérence entre plusieurs répliques. Le coût de la réplication dépend alors non seulement du nombre de répliques mais aussi du temps nécessaire à la propagation des mises à jour. Une synchronisation globale, avec des protocoles appropriés, est souvent nécessaire entre différents nœuds contenant ces répliques (Campelo et al., 2020). En général, de tels protocoles passent difficilement à l'échelle. C'est pour cela que la plupart des stratégies de réplication avec des objectifs de performances ne s'intéressent pas au maintien de la cohérence des données répliquées.

Le problème de la réplication de données est donc un vaste thème de recherche. La proposition de nouvelles stratégies de réplication de données doit être adaptée au contexte des applications tout en prenant en compte le compromis entre des objectifs conflictuels tels que la disponibilité, la cohérence de données et les performances (Pacitti et al., 2005). Dans nos travaux de recherche, nous nous intéressons uniquement à la réplication de données visant l'amélioration des performances dans les systèmes de gestion de données à grande échelle pour des applications décisionnelles OLAP (OnLine Analytical Processing), c'est-à-dire que les données manipulées sont en lecture seule.

1.2 PROBLEMATIQUES ET MOTIVATIONS

De nombreux travaux se sont intéressés à l'amélioration des performances dans les systèmes distribués et parallèles classiques. Dans ce contexte, de nombreuses stratégies de réplication de données ont été proposées. Par la suite, plusieurs travaux ont essayé d'adapter ces stratégies aux systèmes à grande échelle tels que les systèmes de grille de données.

1.2.1 Problèmes liés à la réplication de données dans les systèmes de grille de données

Depuis le début des années 2000, les systèmes de grille de données sont devenus incontournables pour les applications scientifiques nécessitant une importante puissance de calcul et de grandes capacités de stockage (Foster, 2002). Dans ce contexte, la réplication de données permet de garantir une certaine QdS pour les utilisateurs, par exemple en termes de disponibilité de données et de réduction des temps d'accès.

Bien que de nombreuses stratégies de réplication de données ont été proposées dans les systèmes distribués classiques (Benoit and Rehn-Sonigo, 2008), ces stratégies ne sont pas adaptées aux systèmes de grille de données à cause des caractéristiques de ces systèmes, que nous citons ci-dessous.

D'abord, nous retrouvons les caractéristiques fondamentales des systèmes de grille telles que la grande échelle et la dynamicité des nœuds. La grande échelle est caractérisée par (i) un espace de recherche qui dépasse largement celui d'un système distribué classique, (ii) des ressources hautement hétérogènes à différent niveaux (ressources de type matériel, sources de données, réseau de communication ou environnement de développement), (iii) un nombre plus important de sources de données et enfin, (iv) un gros volume de données (Pacitti et al., 2007). La dynamicité signifie qu'un nœud peut rejoindre ou quitter le système à n'importe quel moment (Hameurlain and Morvan, 2009). De plus, le mode d'accès aux données par les utilisateurs change constamment dans de tels systèmes. Il est évident que ces caractéristiques doivent être prises en compte lors de la conception d'une stratégie de réplication dans les grilles de données. Il est alors difficile, voir illusoire, d'adapter les stratégies déjà existantes aux systèmes de grille de données.

Ensuite, les nœuds d'un système de grille de données peuvent être organisés suivant différentes topologies, par exemple une topologie hiérarchique multi-niveaux ou à base de graphe. Définir une stratégie de réplication universelle pour n'importe quelle grille de données n'est pas envisageable du moment qu'une stratégie ne produira pas les mêmes performances pour une architecture de grille ou une autre. Une stratégie de réplication doit alors être conçue en fonction de la topologie pour laquelle elle a été proposée.

Aussi, les objectifs de la réplication de données cités précédemment s'avèrent conflictuels dans les systèmes à grande échelle comme les grilles de données. Ainsi, quand on vise à répliquer les données pour augmenter leur disponibilité, cela se fait naturellement au prix d'importants transferts de données entre nœuds distants. De plus, un bon équilibrage de charge entre les nœuds d'un système n'est pas forcément garantit. Cela a forcément un impact sur les performances de tels systèmes. Il convient alors de trouver un compromis entre différents objectifs.

D'autres problèmes tels que le placement de répliques et la recherche du nombre de répliques ont fait l'objet de nombreux travaux de recherche à cause de leur impact sur les performances. Le placement de données dans les systèmes de grille de données est un problème NP-difficile (Du et al., 2011). Dans ce contexte, de nombreuses stratégies dans la littérature visent à réduire l'espace de recherche, ce qui réduit le temps nécessaire à un tel placement (Mansouri and Javidi, 2017), tandis que d'autres stratégies visent à déterminer un nombre de répliques proche de l'optimal (Muthu and Kumar, 2017).

1.2.2 Problèmes liés à la réplication de données dans les systèmes Cloud

La convergence de deux tendances à savoir la mise à disposition d'applications et la virtualisation de l'infrastructure a donné naissance au paradigme de Cloud Computing (informatique en nuage) que nous décrivons dans le chapitre 3. Cela a conduit à la prolifération de nombreuses solutions évolutives de gestion des données telles que le stockage distribué de fichiers et d'objets (par exemple, HDFS), les bases de données NoSQL (par exemple, MongoDB) et les frameworks de traitement de données massives (par exemple, MapReduce et Spark), à la base d'une riche offre de services de Cloud (IaaS, PaaS,

SaaS, DBaaS, etc.) que nous décrivons également dans le chapitre 3. Dans ce contexte, il existe de nombreux travaux de synthèse sur la gestion de données dans les systèmes Cloud (Chaudhuri, 2012; Ozsu and Valduriez, 2020). Par ailleurs, le succès des Cloud commerciaux qui s'appuient sur des SGBD relationnels prouve que les SGBD relationnels sont toujours importants à l'ère du Cloud, en dépit des critiques visant les SGBD relationnels par rapport à leurs flexibilité et performances (Stonebraker et al., 2010). Comme exemples, nous citons les services Relational Database Service (RDS) d'Amazon[3], Azure SQL Database de Microsoft[4], Cloud SQL de Google[5] ou encore Oracle Database Cloud service d'Oracle[6].

Afin de répondre aux besoins croissants des utilisateurs, appelés locataires, les fournisseurs de Cloud offrent une quantité, en apparence infinie, de ressources, par exemple de stockage et/ou de calcul, disponibles dans différents Centres de Données (CD) (Foster et al., 2008). En effet, de nombreuses organisations exploitent plusieurs CDs géographiquement distribués à travers le monde pour améliorer la latence perçue par l'utilisateur, augmenter la fiabilité de données ou encore diminuer les coûts des ressources (Mansouri and Buyya, 2019; Liu et al., 2020). Les données sont alors naturellement réparties sur des CDs ayant des politiques et des configurations matérielles et logicielles différentes. Par exemple, plusieurs fournisseurs de Cloud proposent le stockage en tant que service (Amazon S3, Google Cloud Storage et Microsoft Azure) dans plusieurs CDs répartis dans le monde. D'autres applications, par exemple le service Web de partage de photos d'Amazon, sont déployées dans plusieurs CDs. Les applications d'aujourd'hui doivent alors être en mesure de fournir à leurs clients un service à faible latence en tirant profit des emplacements distribués pour le stockage. Afin d'atteindre cet objectif, nous nous basons sur la réplication de données dans les systèmes Cloud exploitant des CDs distribués à grande échelle.

En dépit du fait que de nombreuses stratégies de réplication de données ont été proposées dans les systèmes de grille de données, elles sont difficiles à adapter aux systèmes Cloud. Ces stratégies visent à obtenir les meilleures performances du système sans prendre en compte le coût économique de la réplication (Mokadem and Hameurlain, 2020). En effet, une solution courante dans les grilles de données, consiste à maximiser l'utilisation des ressources disponibles afin d'obtenir les meilleures performances possibles. Cela passe par la création d'autant de répliques que possible. Cette solution n'est pas réaliste dans les systèmes Cloud car cela peut entraîner une consommation inutile des ressources et une réduction des bénéfices pour le fournisseur.

Les stratégies proposées dans les systèmes Cloud doivent alors non seulement répondre aux problématiques classiques citées auparavant, à savoir : (i) les conditions de la réplication, (ii) le choix des données à répliquer, (iii) le placement des nouvelles répliques et (iv) le nombre de répliques nécessaires à la satisfaction de la QdS, mais elles doivent également prendre en compte les coûts économiques liés à la réplication tels que le coût engendré par la réplication et la rentabilité du fournisseur. En effet, un fournisseur de Cloud a pour but de générer un profit comme toute entreprise économique en plus d'assurer une certaine QdS pour les locataires. A titre d'exemple, le bénéfice d'exploitation d'AWS (le Cloud d'Amazon)

[3] https://aws.amazon.com/fr/rds/
[4] https://azure.microsoft.com/fr-fr/free/sql-database/
[5] https://cloud.google.com/sql/
[6] https://www.oracle.com/fr/database/cloud-services.html

était de 2.18 milliards de dollars[7], au 4ème trimestre 2019, contre 1.35 milliards de dollars un an plus tôt.

Logiquement, les locataires s'attendent aux meilleures performances possibles au moyen de faibles coûts. Cependant, cela est parfois difficile à fournir et très coûteux pour le fournisseur, surtout lorsque ces performances sont attendues dans un environnement caractérisé par une charge de travail dynamique. En effet, le faible coût et les bonnes performances sont deux objectifs souvent contradictoires. C'est pour cela que la plupart des Cloud commerciaux n'incluent pas les performances, notamment en termes de temps de réponse, dans le SLA (Service Level Agreement), un contrat de niveau service entre le fournisseur et son client (Wu and Buyya, 2010). De ce fait, les stratégies de réplication de données dans les systèmes Cloud doivent alors répondre aux objectifs suivants :

(i) Fournir une QdS fiable aux locataires en satisfaisant le SLA sans sacrifier le profit économique du fournisseur (Buyya et al., 2009), en particulier lorsque ces stratégies sont proposées pour des applications OLAP.

(ii) Permettre une gestion élastique des ressources en ajustant de manière dynamique l'allocation de ces ressources par le fournisseur à ses locataires suivant le modèle de tarification 'pay as you go' (Armbrust et al., 2010), c'est-à-dire que le locataire ne paye au fournisseur que ce qu'il consomme comme ressources.

Dans la littérature, de nombreuses stratégies de réplication se sont intéressées aux problématiques de placement (Kumar et al., 2014) ou de nombre de répliques (He et al., 2018). La plupart de ces stratégies visent l'augmentation de la disponibilité de données (Liu and Shen, 2016; Kaseb et al., 2019) et n'intègrent pas la satisfaction des performances, en termes de temps de réponse, dans le SLA pour les mêmes raisons citées plus haut. De plus, la plupart de ces stratégies (Edwin et al., 2019) visent seulement à réduire la consommation des ressources générée par la réplication (par exemple en termes de stockage et/ou de bande passante du réseau) sans se focaliser sur les coûts économiques liés à la réplication. Dans ce contexte, certains travaux sont mentionnés comme prenant en compte le coût économique de la réplication. Néanmoins, le coût considéré n'est pas nécessairement un coût monétaire. Par exemple, ce coût est considéré comme une valeur budgétaire initialement assignée pour les CDs dans la stratégie proposé dans (Gill and Singh, 2016).

Seules certaines stratégies s'intéressent à la satisfaction des objectifs de performances pour les locataires tout en prenant en compte les coûts économiques engendrés par la réplication et le profit économique (monétaire) du fournisseur (Wu et al., 2013; Zeng et al., 2016; Casas et al., 2017; Liu and Shen, 2017; Mansouri and Buyya, 2019). De plus, ces stratégies de réplication de données envisagent souvent une réplication par requête, qui répond systématiquement à toute variation de la charge de travail ou à un changement de popularité des données. Cela est fait aux dépens de temps de réponses élevés et des frais généraux supplémentaires. Aussi, les travaux qui ciblent un SGBD relationnel opérant dans le Cloud sont significativement réduits. Il convient alors de proposer des stratégies de réplication de données qui répondent au compromis entre les performances pour les locataires et le profit économique pour le fournisseur.

[7] ZDnet.fr (mai 2020)

1.3 CONTRIBUTIONS

Depuis mon recrutement en tant que Maitre de Conférences à l'UPS Toulouse 3 en septembre 2007, mes travaux de recherche ont été centrés autour de l'évaluation et de l'optimisation de requêtes réparties en environnements à grande échelle. Des thématiques variées ont été visitées :

(i) J'ai commencé par l'évaluation de l'impact des signatures algébriques (Litwin et al., 2007) sur les performances de l'opérateur de jointure (Mokadem et al., 2008a; 2008b et 2010a),

(ii) Ensuite, je me suis intéressé aux problèmes de : (a) la découverte de ressources dans les grilles de données hiérarchiques (Mokadem et al., 2010b; 2011; 2012a et 2012b) et, (b) la découverte de sources de données en tenant compte de la sémantique dans les environnements de grille de données (Ketata et al., 2010; 2011a; 2011b et 2011c; Benslimane et al., 2013; Mokadem et al, 2013). Une partie de ces travaux a été menée dans le cadre du projet ANR PAIRSE que nous décrivons en Annexe A.

(iii) Depuis 2013, mes travaux de recherche sont focalisés sur les problématiques liées à la réplication de données dans les systèmes de gestion de données à grande échelle.

Un certain nombre de contributions ont été proposées. Elles ont été réalisées dans le cadre de l'encadrement ou co-encadrement de huit doctorants. Quatre doctorants (dont deux à l'UPS) ont soutenu leur thèse : Imen Ketata (Ketata, 2012), Uras Tos (Tos, 2017), Said Limam (Limam, 2018) et khaoula Tabet (Tabet, 2018) et quatre autres doctorants (dont un à l'UPS) sont encore en cours de thèse (Morgan Séguéla, Amel Khelifa, Abdenour Lazeb et Hagamalala Santatra Bernardin).

Pour des raisons de cohérence, je ne présente ici que les travaux de recherche portant sur la réplication de données dans les systèmes de grille de données et dans les systèmes Cloud. Dans ce qui suit, nous citons les principales contributions issues de ces travaux. Ces contributions sont détaillées dans les chapitres 2, 3 et 4.

1.3.1 Classification des stratégies de réplication de données dans les systèmes de grille de données

L'étude de l'état de l'art relatif aux stratégies de réplication dans les grilles de données a montré que la plupart des travaux de synthèse ont classé ces stratégies en se basant sur les critères suivants : (i) la nature de la réplication (stratégies statiques vs. dynamiques) (Mansouri and Javidi, 2019), (ii) le mécanisme de contrôle (stratégies centralisées vs. décentralisées) (Ma et al., 2013), ou encore (iii) l'origine de la réplication (stratégies initiées par le serveur vs. initiées par le client (Van Steen and Pierre, 2010). Nous avons proposé une nouvelle classification de ces stratégies en se basant sur deux autres critères :

Le premier critère sur lequel se base notre classification est la fonction objectif visée par une stratégie de réplication (Mokadem and Hameurlain, 2015). Dans de nombreuses stratégies, définir une fonction objectif revient à privilégier la localité de données en plaçant les répliques des données populaires aussi près que possible des utilisateurs. D'autres stratégies privilégient le placement de nouvelles répliques sur des nœuds disposant d'une meilleure bande passante du réseau avec les nœuds qui demandent ces données. Il est alors évident qu'une stratégie de réplication est définie suivant la fonction objectif pour laquelle elle a été

conçue. Or, très peu de travaux (Goel and Buyya, 2006) exploitent (partiellement) la fonction objectif comme un critère de classification de ces stratégies. Nous avons alors classé ces stratégies comme suit : (a) les stratégies qui visent à améliorer la localité de données, du point de vue temporel, géographique ou spatial (Ranganathan and Foster, 2001), (b) les stratégies qui visent à améliorer la disponibilité de la bande passante du réseau (Park et al., 2004), (c) les stratégies qui visent à réduire les coûts de la réplication en se basant sur un modèle de coûts (Muthu and Kumar, 2017), et (d) les stratégies qui exploitent des comportements utilisés dans le commerce (par exemple, en considérant les données comme des biens échangeables) afin de réduire les coûts de la réplication (Andronikou et al., 2012).

Nous pensons également qu'il est important de classer ces stratégies suivant le type d'architecture de la grille pour laquelle elles ont été conçues, vu que la topologie d'une grille affecte significativement les performances de ces stratégies (Tos et al., 2015). Nous retrouvons les stratégies proposées pour : (a) les architectures hiérarchiques multi-niveaux (Cui et al., 2015), (b) les architectures hiérarchiques du point de vue de la bande passante (Mansouri et al., 2013), (c) les architectures Pair à Pair (P2P) (Chettaoui and Ben Cherrada, 2014), (d) les architectures à base de graphe (Souravlas and Sifaleras, 2017b) et (e) les architectures hybrides (Azari et al., 2018).

Nous avons évalué l'impact de ces critères de classification sur les performances en utilisant le simulateur OptorSim (Bell et al., 2003b). Nos expériences ont montré que la satisfaction d'une seule fonction objectif, comme c'est le cas dans la plupart des stratégies existantes, peut conduire à des performances dégradées. Il est alors préférable qu'une stratégie vise la recherche d'un compromis entre plusieurs fonctions objectif (Mokadem and Hameurlain, 2015). D'un autre coté, l'évaluation des performances a montré qu'une stratégie conçue pour une grille de données à base de graphe, bien que plus réaliste, nécessite un coût de maintenance élevé de l'infrastructure à cause de la dynamicité des nœuds. Ceci impacte négativement les performances. En revanche, une stratégie conçue pour une architecture hybride permet d'aboutir à de meilleures performances en tirant profit des avantages de différentes architectures, par exemple une bonne disponibilité comme avantage de la topologie hiérarchique et le passage à l'échelle comme avantage de la topologie P2P (Tos et al., 2015).

1.3.2 Classification des stratégies de réplication de données dans les systèmes Cloud

Après avoir étudié les problèmes liés à la réplication de données dans les systèmes Cloud, nous avons passé en revue l'état de l'art relatif aux stratégies de réplication de données dans de tels systèmes. Nous avons constaté que la plupart des travaux de synthèse dans la littérature ont classé ces stratégies comme étant (i) statiques vs. dynamiques (Milani and Navimipour, 2016), et/ou (ii) centralisées vs. décentralisées (Gopinath and Sherly, 2018a). En se basant sur d'autres critères, parfois spécifiques aux environnements Cloud, nous avons proposé une nouvelle classification de ces stratégies selon : (i) l'orientation du bénéfice, puisque les stratégies existantes visent soit à augmenter le profit du fournisseur ou à réduire le coût pour les locataires (Tos et al., 2018), (ii) la fonction objectif visée par une stratégie de réplication de données (par exemple l'amélioration de la disponibilité des données ou de la bande passante du réseau), (iii) le nombre d'objectifs visés par une stratégie, vu que certaines stratégies visent la satisfaction d'un seul objectif tandis que d'autres prennent en compte le

compromis entre plusieurs objectifs (Mokadem and Hameurlain, 2020), (iv) la nature de l'environnement pour lequel une stratégie a été proposée, notamment en ce qui concerne le nombre de fournisseurs de services (Mokadem et al., 2020), et (v) la prise en compte des coûts économiques liés à la réplication, par exemple le profit économique, notamment monétaire, pour le fournisseur (Tos et al., 2016) et la consommation énergétique du fournisseur lors de la réplication (Séguéla et al., 2019a).

Nous avons également évalué l'impact de ces critères sur les performances des stratégies de réplication en utilisant l'outil de simulation CloudSim (Calheiros et al. 2010). Nos expériences ont montré qu'une stratégie qui vise le compromis entre plusieurs objectifs pour les locataires produit de meilleures performances. Afin d'évaluer la profitabilité de la réplication pour le fournisseur, les dépenses de ce dernier doivent aussi inclure les coûts économiques liés à la réplication, par exemple les coûts engendrés par la réplication (Mokadem and Hameurlain, 2020).

1.3.3 Gestion de la réplication de données dans les systèmes Cloud

Nous avons mis en évidence les principes directeurs de la conception de stratégies dynamiques de réplication de données dans les systèmes Cloud. Dans ce contexte, la proposition de stratégies efficaces permettant de prendre en compte le compromis entre la garantie d'une QdS (en termes de performances) pour les locataires et la rentabilité pour le fournisseur, constitue un défi majeur (Tos et al., 2018).

Dans cette perspective, nous avons répondu aux problématiques liées à la réplication de données dans les systèmes Cloud exploitant des CDs distribués à grande échelle. La prise de décision de réplication est dirigée par la satisfaction du SLA. Elle est basée sur des seuils, en termes de temps de réponse. Avant l'exécution de chaque requête d'un locataire, nous avons estimé le temps de réponse d'un plan d'exécution parallèle fourni par le SGBD. Nous avons considéré les plans linéaires gauches, linéaires droits ainsi que les plans ramifiés (bushy). Pour cela, nous nous sommes basés sur les résultats fondamentaux obtenus dans les systèmes parallèles et distribués classiques, en exploitant notamment les différentes formes de parallélisme (intra-opération, indépendant et pipeline). Une réplication de données est envisagée si et seulement si une violation du SLA est constatée, notamment en termes d'objectif de temps de réponse pour le locataire. De plus, la création d'une réplique est considérée, le plus souvent, par groupe de requêtes afin de réduire les coûts liés à la réplication (Mokadem and Hameurlain, 2020).

Nous avons proposé une heuristique pour le placement de nouvelles répliques en exploitant la localité du point de vue de la bande passante du réseau, ce qui permet de réduire l'espace de recherche. Cette heuristique vise à trouver un emplacement acceptable avec des coûts réduits de réplication. L'objectif n'est pas de trouver un emplacement optimal mais de produire des performances acceptables qui satisfont le contrat SLA, en termes de temps de réponse, tout en étant profitable pour le fournisseur. La rentabilité du fournisseur constitue alors un autre objectif à satisfaire. En d'autres termes, une nouvelle réplique est réellement créée si et seulement si la réplication est profitable du point de vue économique pour le fournisseur. Pour cela, nous avons étendu les modèles de coûts classiques en intégrant les coûts économiques liés à l'exécution des requêtes des locataires. Dans ce modèle de coûts, qu'on a appelé 'modèle de coûts économique', l'estimation des dépenses du fournisseur inclut également les coûts engendrés par la réplication ainsi que les éventuelles

pénalités payées par le fournisseur à ses locataires en cas de violation du SLA. Dans ce contexte, nous avons développé un algorithme qui permet de limiter le payement répétitif de pénalités par le fournisseur à ses locataires, ce qui augmente le profit économique du fournisseur.

D'un autre coté, le fournisseur doit absolument éviter la création de répliques inutiles lorsque la charge de travail diminue. Pour cela, nous avons proposé un autre algorithme permettant d'ajuster dynamiquement le nombre de répliques afin de permettre une gestion élastique des ressources. Ceci permet au fournisseur de s'adapter à la demande des locataires (Tos et al., 2018).

Enfin, nous avons comparé les performances de nos propositions à celles d'autres stratégies de réplication de données proposées dans les systèmes Cloud. Pour cela, nous avons étendu l'outil de simulation CloudSim puis, nous avons implémenté l'ensemble des stratégies comparées. L'analyse des résultats a montré que nos propositions ne se contentent pas seulement de satisfaire les objectifs des locataires. Elles prennent en compte le profit économique (monétaire) du fournisseur. De plus, elles produisent de meilleures performances par rapport aux stratégies comparées, dans des conditions défavorables telles que des taux importants d'arrivée de requêtes, des seuils de temps de réponse stricts et des requêtes complexes.

1.4 ORGANISATION DU DOCUMENT

Le reste de ce document est organisé comme suit : Le chapitre 2 dresse un état de l'art relatif aux principales stratégies de réplication de données visant à améliorer la QdS des applications OLAP dans les systèmes de grille de données. Il inclut également une proposition d'une classification de ces stratégies dans de tels systèmes. Le chapitre 3 présente un état de l'art concernant les stratégies de réplication de données dans les systèmes Cloud. Dans ce chapitre, nous proposons également une classification de ces stratégies dans ces systèmes. Le chapitre 4 présente nos principales contributions concernant la réplication de données lors de l'interrogation des bases de données dans les systèmes Cloud. Le chapitre 5 confirme la validité de nos propositions à travers une étude expérimentale. Le chapitre 6 conclut nos travaux de recherche. Nous présentons également quelques nouvelles directions de recherche dans le prolongement de nos propositions. Après la partie bibliographique, mon CV académique est mis en Annexe A. Il synthétise et résume les projets et encadrements auxquels j'ai participé ainsi que les principales responsabilités assurées durant ces dernières années. Enfin, la liste des principales publications issues de mes travaux de recherche est fournie en Annexe B.

CHAPITRE 2

Réplication de données dans les systèmes de grille de données

Résumé

Dans ce chapitre, nous commençons par mettre en évidence les caractéristiques clés des systèmes de grille. Ensuite, nous présentons les principales stratégies de réplication de données dans les systèmes de grille de données à travers les classifications existantes de ces stratégies. Enfin, nous proposons une nouvelle classification de ces stratégies dans les systèmes de grille de données.

Sommaire

2.1 INTRODUCTION

Dans les systèmes de gestion de données à grande échelle, un accès fréquent à ces données réduit sensiblement la bande passante du réseau, ce qui rallonge le temps de transfert de données. Les performances du système sont alors dégradées. D'autre part, placer des répliques de ces données sur l'ensemble des nœuds du système est coûteux et tout simplement irréaliste à cause des contraintes de capacité du réseau et de ressources de stockage limitées. Un placement stratégique de ces répliques permet d'augmenter la disponibilité de données, d'assurer une meilleure tolérance aux pannes, et d'améliorer les performances d'accès à travers notamment la réduction des coûts de transfert de données, l'équilibrage de charge ou encore l'exploitation du parallélisme.

De nombreux systèmes de gestion de données reposent sur la réplication de données afin d'améliorer la QdS pour les utilisateurs, par exemple en termes de performances,

spécialement lorsqu'il s'agit de répondre aux attentes d'applications OLAP manipulant des données en lecture seule. Lorsque les données sont mises à jour, les modifications doivent être propagées à toutes les répliques du système afin de garantir une certaine cohérence des données, ce qui a un impact direct sur les performances. Plusieurs modèles et protocoles de cohérence ont été proposés pour la propagation de ces mises à jour (Pacitti et al., 2005; Campelo et al., 2020). Généralement, ces protocoles passent difficilement à l'échelle. En effet, le théorème CAP stipule qu'il n'est pas possible d'offrir à la fois cohérence, disponibilité et tolérance au partitionnement (Brewer, 2000). Il est alors parfois nécessaire de faire un compromis sur l'une de ces trois propriétés en fonction des contraintes de ces applications. Dans le reste de ce manuscrit, on ne s'intéresse qu'aux stratégies de réplication de données qui ont pour objectif l'amélioration des performances tout en manipulant des données en lecture seule.

Dans la littérature, de nombreuses stratégies de réplication de données ont été proposées dans les systèmes traditionnels tels que : (i) les SGBD distribués (Kemme et al., 2010) incluant les SGBD commerciaux (Par exemple, Oracle), (ii) les systèmes distribués et parallèles classiques (Sidel et al., 1996; Benoit and Rehn-Sonigo, 2008; Zhao et al., 2017; He and Sun, 2018) et (iii) les systèmes mobiles (Tu et al., 2006; Guerrero-Contreras et al., 2015). Elles permettent notamment, de répondre aux problématiques suivantes : (i) Quelles données doivent être répliquées ? (ii) Quand la réplication doit-elle avoir lieu ? (iii) Où placer les nouvelles répliques ? Et enfin, (iv) Combien de répliques doivent être créées ?

Dans les prochaines sections, nous verrons que les stratégies proposées dans les systèmes traditionnels cités ci-dessus ne sont pas adaptées aux systèmes à grande échelle tels que les systèmes de grille de données (Tos et al., 2015) et les systèmes P2P (Spaho et al., 2015). Nous nous intéressons à la réplication de données dans les systèmes de grille de données. D'abord, nous mettons en évidence les concepts de base et les caractéristiques de tels systèmes. Ces caractéristiques ont un impact important sur la conception d'une stratégie efficace de réplication de données. Nous passons en revue les nouvelles problématiques liées à la réplication de données dans ces systèmes. Ensuite, nous présentons un large état de l'art incluant les principales stratégies de réplication de données proposées dans la littérature. Les classifications existantes de ces stratégies sont discutées. Enfin, nous nous basons sur deux critères pour la proposition d'une nouvelle classification de ces stratégies dans les systèmes de grille de données.

2.2 LES GRILLES DE DONNEES : CONCEPTS DE BASE

2.2.1 Contexte et définitions

Face aux besoins des applications informatiques concernant le stockage et le traitement de grandes masses de données, de nouvelles infrastructures ont vu le jour durant les années 90. Son inspiration vient de la fiabilité et de la facilité d'utilisation du réseau de distribution électrique. La transposition de ce concept au monde de l'informatique a donné naissance à la notion de calcul en grille (Grid Computing).

Dans (Chervenak et al., 2000), une grille informatique est définie comme une infrastructure matérielle et logicielle dont le but est d'interconnecter des ressources partagées, distribuées et hétérogènes afin de répondre aux besoins des applications scientifiques caractérisées par un calcul intensif et de grands volumes de données (Foster, 2001). L'idée est de répartir, de

manière transparente, des calculs et/ou des données provenant de nœuds distribués à grande échelle. Cela se fait à travers la mutualisation des ressources de calcul, de stockage ou de bande passante du réseau (Pacitti et al., 2007). Selon (Foster, 2002), une grille peut être vue comme un ensemble d'Organisations Virtuelles (OVs). Chaque OV est dédiée à un domaine d'application (centre d'intérêt) et met un ensemble de ressources à disposition de ses utilisateurs. Trois critères sont cités pour caractériser une grille : (i) le contrôle décentralisé des ressources (ressources autonomes), ce qui permet un meilleur passage à l'échelle (scalabilité), (ii) la nécessité de se baser sur des standards, par exemple l'architecture OGSA (Open Grid Service Architecture) (Foster et al., 2004) est proposée pour les applications qui accèdent à des services Web et (iii) la délivrance d'une QdS, par exemple en termes de temps de réponse, de disponibilité et de sécurité.

2.2.2 Grilles de données

D'un point de vue de services qu'elles offrent, les grilles ont commencé d'abord à exploiter la puissance de calcul des nœuds présents sur le réseau, par exemple Internet. On parle alors de *grilles de calcul*, utilisées pour des applications nécessitant des calculs intensifs (Bolze et al., 2006). On retrouve également les *grilles d'informations* permettant le partage d'informations à travers un réseau. Le Web peut être vu comme une grille d'informations. Enfin, nous retrouvons *les grilles de données* qui diffèrent des grilles de calcul du fait de la manipulation de données plus volumineuses, hétérogènes et réparties dans plusieurs OVs (Venugopal et al., 2006). Dans ce chapitre, nous nous intéressons à la réplication de données dans les environnements de grille de données.

Une grille de données est une infrastructure de grille spécialisée qui fournit une solution de gestion de données évolutive pour les expériences scientifiques ou applications intensives générant un gros volume de données (Chervenak et al., 2000). De nombreux domaines de recherche comme la physique (Takefusa et al., 2003), l'astronomie (Deelman et al., 2002), la biologie (Maltsev et al., 2006) et la climatologie (Chervenak et al., 2003) se sont basés sur les grilles de données. A titre d'exemple, plus de 150 organisations à travers le monde participent au projet d'accélérateur de particules LHC (Large Hadron Collider), démarré en 2007 au sein du centre de physique des particules CERN[8]. Il produit plus de 15 péta-octets de données par an, qui sont accédées par des physiciens du monde entier (Chang and Chen, 2007).

Une gestion de données sur une telle infrastructure pose de nombreux problèmes et constitue un réel défi : découverte et allocation de ressources, efficacité et confidentialité d'accès, disponibilité des ressources, mécanisme de cache, gestion de la réplication, modèle de coûts, benchmarking,…etc. Les caractéristiques telles que la dynamicité de nœuds, l'hétérogénéité de données et des ressources, et la répartition à grande échelle complexifient les problèmes cités ci-dessus. La dynamicité signifie qu'un nœud peut rejoindre ou quitter le système à n'importe quel moment. D'un autre coté, la grande échelle signifie : (i) la présence d'un grand nombre de sources de données et de calcul, (ii) des ressources de calcul hétérogènes, (iii) des ressources très nombreuses, générant un grand volume de données et enfin, (iv) l'interconnexion entre ces ressources se fait à travers des bandes passantes hétérogènes, parfois faibles (Pacitti et al., 2007; Hameurlain and Morvan, 2009). En effet, l'hétérogénéité des ressources concerne aussi bien les sources de données, les ressources de

[8] http://www.lhc-france.fr/

calcul ou encore les ressources de type RAM, bande passante E/S ou de réseau. Dans ce qui suit, nous nous intéressons aux problèmes liés à la réplication de données dans les systèmes de grille de données.

2.3 PROBLEMES LIES A LA REPLICATION DE DONNEES DANS LES SYSTEMES DE GRILLE DE DONNEES

Les applications ayant recours à des systèmes de gestion de données à grande échelle, par exemple les grilles de données, manipulent généralement un gigantesque volume de données (Venugopal et al., 2006). Dans de tels systèmes, les données générées par des expériences scientifiques au sein d'une institution sont souvent traitées par d'autres institutions situées dans d'autres régions du monde pour être expédiées aux demandeurs.

Un accès fréquent à ces données, associé à une faible bande passante, peut entraîner des délais importants. Par ailleurs, la panne d'un nœud contenant la seule copie d'une unité de données rend impossible la récupération de celle-ci. L'amélioration de la disponibilité des données et la réduction du temps d'accès à ces données constituent alors des objectifs à atteindre à travers la conception d'une stratégie de réplication de données.

Nous avons mis en évidence, dans la section précédente, quelques caractéristiques spécifiques à l'infrastructure de grille de données, telles que la dynamicité des nœuds et la grande échelle (Pacitti et al., 2007). Ces caractéristiques introduisent de nouvelles problématiques lors de la conception d'une stratégie de réplication de données. Sans la prise en compte de ces caractéristiques, un accès fréquent à des données réparties à grande échelle entraînerait une dégradation des performances. D'autres caractéristiques des grilles de données, citées ci-dessous, font que les stratégies proposées pour des systèmes distribués classiques s'avèrent insuffisantes dans les systèmes de grille de données.

D'abord, différentes architectures ont été proposées pour les grilles de données, par exemple les grilles de données dont la topologie est hiérarchique ou à base de graphe. Il n'est donc pas possible de définir une stratégie de réplication universelle valable dans n'importe quelle topologie. Ensuite, le comportement des utilisateurs change souvent dans les environnements de grille de données. Les stratégies de réplication de données doivent alors prendre en compte la variation dans le temps des modes d'accès aux données.

D'un autre coté, l'obtention d'une configuration de réplication optimale sur l'ensemble d'un système de gestion de données à grande échelle est un problème NP-difficile (Tang et Xu, 2005). Dans ce contexte, la granularité d'une réplication de données est un aspect important lors de la conception d'une stratégie de réplication de données. Idéalement, une stratégie de réplication doit s'adapter à toute granularité de données. Toutefois, une réplication visant l'obtention de bonnes performances nécessite de choisir une bonne granularité des données. En effet, les stratégies de réplication fournissent des performances différentes lorsque différentes granularités sont appliquées (Van Steen and Pierre, 2010). Dans la littérature, on retrouve trois niveaux de granularité de données : (i) des fichiers individuels (Kunszt et al., 2005), (ii) plusieurs fichiers en même temps (granularité de collections) (Garcia-Carballeira et al., 2007) ou encore (iii) des petites subdivisions, par exemple des relations ou des fragments d'une relation (Van Steen and Pierre, 2010). Dans les solutions que nous proposons dans ce manuscrit, un fragment d'une relation (qu'on appellera relation) constitue la granularité de réplication.

Une autre caractéristique spécifique aux grilles de données réside dans le fait que la grille met à disposition de nombreuses ressources disponibles au sein de plusieurs institutions. Les stratégies de réplication de données dans de tels environnements profitent alors au mieux de ces ressources. Elles visent à obtenir les meilleures performances possibles en réduisant au mieux les temps d'accès aux données. L'utilisation de ces ressources est alors maximisée dans le but de répondre, le mieux possible, aux exigences des applications gourmandes en ressources.

Au vu de toutes les caractéristiques citées ci-dessus, les problématiques liées à la réplication de données deviennent encore plus difficiles à résoudre :

(i) Il est important d'établir des critères spécifiques permettant le choix de l'entité à répliquer à cause du nombre élevé des sources de données et du volume important de données.

(ii) La décision de répliquer les données est difficile à prendre, d'autant plus que le comportement d'un utilisateur change constamment dans un système de grille. Il est alors nécessaire que la décision de création/suppression d'une réplique puisse intervenir au bon moment et de manière dynamique.

(iii) Le placement de répliques dans un environnement de grille est une tâche complexe à cause du nombre important de nœuds. En général, placer des répliques près des clients qui les accèdent peut améliorer les performances globales. On notera également l'utilisation d'heuristiques permettant de réduire l'espace de recherche.

(iv) La recherche du nombre optimal de répliques est généralement très coûteuse en termes de ressources (Saadat and Rahmani, 2012). La recherche d'une solution satisfaisante, au mieux proche de l'optimal, est alors plus intéressante.

Enfin, les stratégies de réplication proposées dans la littérature répondent différemment aux problématiques ci-dessus (Ranganathan et Foster, 2001; Park et al., 2004; Tang et al., 2005; Chang and Chang, 2008; Andronikou et al., 2012; Cui et al., 2015; Nazir et al., 2018). Certaines d'entre elles visent la satisfaction d'un seul objectif pour les utilisateurs, par exemple en termes de disponibilité des données, avec un minimum de conséquences indésirables sur les autres exigences, par exemple les performances en termes de temps de réponse. D'autres stratégies visent à satisfaire plusieurs objectifs, souvent étroitement liés et clairement définis dès la conception. Elles tentent de prendre en compte le compromis entre ces différents objectifs. En effet, certains objectifs s'avèrent conflictuels dans les systèmes à grande échelle. Par exemple, la disponibilité de données et la réduction de la consommation de bande passante sont parfois conflictuels au regard du grand volume de données manipulé dans de tels systèmes. Dans ce qui suit, nous nous concentrons sur les stratégies de réplication visant à améliorer les performances d'un système de grille de données.

2.4 CLASSIFICATION PROPOSEE DES STRATEGIES DE REPLICATION DANS LES SYSTEMES DE GRILLE DE DONNEES

Dans la littérature, de nombreuses stratégies de réplication de données ont été proposées dans les systèmes de grille de données. Par ailleurs, la plupart des travaux de synthèse ont classé ces stratégies en : (i) stratégies statiques vs. dynamiques (Chervenak et al., 2002; Cibej et al., 2005; Mansouri and Javidi, 2019) et/ou stratégies centralisées vs. décentralisées (Dogan,

2009 ; Amjad et al., 2012; Ma et al., 2013). D'autres études considèrent d'autres critères : (iii) l'initiation de la réplication (réplication initiée par le serveur vs. par le client) (Nicholson et al., 2008; Chervenak et al., 2008; Dogan, 2009 ; Van Steen and Pierre, 2010) ou encore (iv) l'existence d'une condition pour la création d'une réplique (Al Mistarihi and Yong, 2008).

Stratégies statiques vs. dynamiques. Dans les stratégies statiques de réplication de données, toutes les décisions de réplication sont prises avant que le système ne soit opérationnel. Ainsi, l'emplacement des répliques et le nombre de répliques sont fixés à l'avance (Chervenak et al., 2002; Tatebe et al., 2002; Bell et al., 2003a; Fu et al., 2013; Souravlas and Sifaleras, 2017a). Ensuite, une réplique reste persistante jusqu'à ce qu'elle soit supprimée par un utilisateur ou que sa durée de vie expire. Ces stratégies produisent de meilleurs résultats lorsque les ressources du système sont stables. Cependant, le comportement des utilisateurs et la bande passante du réseau varient dans le temps, par exemple entre la nuit et le jour. Ces stratégies ne s'adaptent pas à ces situations. En effet, répliquer avec la même fréquence, quand la charge de travail change, impacte négativement les performances du système.

Les stratégies dynamiques (Lamehamedi et al., 2002; Park et al., 2004; Tang et al., 2005; Chang and Chang, 2008; Mansouri and Dastghaibyfard, 2012; Mansouri and Javidi, 2018c) permettent de surmonter ces problèmes. Des répliques sont créées, placées et gérées de manière dynamique en fonction des modes d'accès des utilisateurs et de la disponibilité des ressources, par exemple en termes de stockage, tandis que les répliques inutiles sont retirées. Ces stratégies sont considérées comme plus appropriées pour les systèmes de grille de données en raison des caractéristiques de ces systèmes. C'est pour cela que la plupart des stratégies de réplication proposées dans de tels systèmes sont dynamiques.

Stratégies de réplication centralisées vs. décentralisées. Certain travaux (Dogan, 2009; Sashi and Thanamani, 2011; Mansouri and Dastghaibyfard, 2012, Amjad et al., 2012; Ma et al., 2013) se sont basés sur la nature de la prise de décision pour classer les stratégies de réplication de données. Cette classification concerne principalement les stratégies dynamiques qui peuvent être mises en œuvre de manière centralisée ou décentralisée.

Dans les stratégies de réplication centralisées (Tang et al., 2005; Lei et al., 2008; Chang and Chang, 2008; Lin et al., 2008; Rasool et al., 2009; Bsoul et al., 2010; Andronikou et al., 2012), un seul nœud est responsable de la gestion de la réplication en collectant toutes les informations. Dans (Rasool et al., 2009), la décision de réplication est gérée par une autorité centrale appelée GRS (*Grid Replication Scheduler*). GRS cible les fichiers dont la fréquence d'accès est supérieure à la moyenne et crée des répliques sur le parent du client qui génère le plus de demandes. Ceci conduit à une latence d'accès et une charge importante sur le nœud central, qui peut devenir un goulet d'étranglement. L'approche centralisée est donc plus facile à mettre en œuvre. Une seule entité est responsable de toutes les décisions et a connaissance de chaque aspect du système. Néanmoins, cette entité peut constituer un point de défaillance si elle tombe en panne.

Dans les stratégies de réplication décentralisées (Ranganathan and Foster, 2001; Lei et al., 2008; Sashi and Thanamani, 2011; Muthu and Kumar, 2017; Omer and Abdella, 2018; Azari et al., 2018), on ne retrouve plus de nœud central. Les nœuds eux-mêmes décident du moment de la réplication. Le système peut alors envisager une panne ou la déconnexion de certains nœuds. Néanmoins, une synchronisation est nécessaire afin d'obtenir de meilleurs résultats.

De plus, l'absence d'informations sur l'état global du système peut engendrer la création de répliques inutiles (Ranganathan et al., 2002).

Réplication initiée par le serveur vs. initiée par le client. Cette classification considère l'origine de la réplication de données (Nicholson et al., 2008; Chervenak et al., 2008; Dogan, 2009 ; Van Steen and Pierre, 2010; Mokadem and Hameurlain, 2015). Le serveur correspond au nœud qui héberge les données et qui décide de créer une réplique, tandis que le client correspond au nœud qui demande les données puis les met dans son stockage local. Dans une réplication initiée par le serveur (Rahman et al., 2008), celui-ci reçoit les demandes d'un certain nombre de clients. Il a donc besoin d'informations suffisantes sur l'état du système pour pouvoir déclencher la réplication. Ce type de réplication est appelé aussi '*Push-Initiated*'. En revanche, une réplication initiée par le client, appelée aussi '*Pull-Initiated*', est réalisée à la demande du client qui décide de stocker temporairement les données en local (Nukarapu et al., 2011). La stratégie de mise en cache simple coté client (Ranganathan and Foster, 2001), constitue un exemple de réplication initiée par le client.

Réplication conditionnelle vs. inconditionnelle. Quelques travaux (Al Mistarihi and Yong, 2008) ont classé les stratégies de réplication en fonction de l'existence d'une condition pour la création d'une réplique. La réplication inconditionnelle consiste à effectuer une réplication à chaque requête, c'est-à-dire que le nœud qui demande un fichier stocke toujours une copie en local. Dans ce contexte, deux algorithmes de réplication de données ont émergé : (i) l'algorithme LRU (*Last Recently Used*) et (ii) l'algorithme LFU (*Last Frequently Used*) (Arlitt et al., 2000). Dans LRU, un nœud réplique toujours les données requises et les met en cache. Ensuite, si la capacité de stockage en local est atteinte, la réplique la plus ancienne est supprimée pour libérer de l'espace. D'un autre coté, LFU fonctionne comme LRU à la différence que la réplique la moins accédée du stockage local est supprimée même si la réplique est nouvellement créée. Cependant, cela peut engendrer des réplications inutiles. Il est à signaler que la plupart des stratégies proposées dans la littérature créent une nouvelle réplique à la suite de la satisfaction d'une ou plusieurs conditions, telles que le dépassement d'un seuil de popularité pour les données.

Dans ce qui suit, nous nous basons sur deux autres critères pour la proposition d'une nouvelle classification de ces stratégies : (i) la fonction objectif pour laquelle une stratégie a été conçue (Mokadem and Hameurlain, 2015) et (ii) le type d'architecture de la grille de données pour laquelle une stratégie a été élaborée (Tos et al., 2015).

2.4.1 Classification suivant la fonction objectif visée

La plupart des stratégies de réplication de données qui visent à améliorer les performances d'un système, par exemple en termes de temps de réponse, essayent de minimiser/maximiser la valeur d'une métrique à travers un comportement qui impacte ses performances (Sivasubramanian et al., 2004). On parle alors de la satisfaction d'une fonction objectif. Dans ce qui suit, nous proposons de classer ces stratégies suivant la fonction objectif visée par chaque stratégie (Mokadem and Hameurlain, 2015). Nous distinguons les stratégies qui visent à : (i) améliorer la localité de données, (ii) améliorer la disponibilité de la bande passante du réseau, (iii) réduire les coûts de réplication en se basant sur un modèle de coûts, ou (iv) réduire les coûts de réplication en se basant sur des comportements économiques.

2.4.1.1 Stratégies de réplication de données améliorant la localité de données

Ces stratégies sont adaptées aux requêtes des utilisateurs. La plupart d'entre elles visent à maximiser la localité des données en exploitant la popularité de ces données (Tang et al., 2005). La popularité de données représente le taux d'accès à ces données et constitue un paramètre que de nombreuses stratégies utilisent. L'exploitation de la popularité de données pour la réplication a été initialement proposée par (Gwertzman and Seltzer, 1995) en répliquant les données les plus demandées. Dans les stratégies proposées dans (Ananthanarayanan et al., 2011; Mansouri and Javidi, 2018c), la popularité d'une relation *R* est déterminée en analysant l'accès des utilisateurs à *R* pendant une unité de temps. Les auteurs dans (Lei et al., 2008) proposent d'utiliser les historiques d'accès aux données afin de calculer rapidement leurs popularités. Dans la stratégie proposée dans (Shorfuzzaman et al., 2009), une heuristique est utilisée afin de déterminer les données populaires alors que d'autres travaux (Lee et al., 2012) appliquent un relevé périodique des accès aux données afin de déterminer leur popularité.

Les auteurs dans (Ranganathan and Foster, 2001) ont proposé cinq stratégies permettant d'améliorer la localité des données. La plus connue est la stratégie du meilleur client (*Best Client*). Dans cette stratégie, chaque nœud enregistre l'historique des demandes pour ses fichiers. Si la popularité d'un fichier dépasse un certain seuil, une réplique est créée dans le nœud qui a le plus grand nombre de demandes de ce fichier. Ce nœud correspond alors au meilleur client pour ce fichier. Les auteurs ont introduit trois différents types de localité, à savoir :

(i) Une localité temporelle qui signifie que les données accédées récemment ont une grande probabilité d'être de nouveau demandée (sous peu),

(ii) Une localité géographique qui signifie que les données accédées récemment par un client doivent probablement être demandées par un client adjacent, et

(iii) Une localité spatiale qui signifie que les données proches de données accédées récemment ont une grande probabilité d'être accédées dans un proche avenir.

Par ailleurs, trois modes d'accès ont été expérimentés dans les travaux de (Ranganathan and Foster, 2001) : (i) un accès aléatoire, (ii) un accès avec une localité temporelle et (iii) un accès avec une localité temporelle en plus d'une localité géographique réduite. Les auteurs ont affirmé que différents modes d'accès nécessitent différentes stratégies de réplication. Une réduction significative de la consommation de la bande passante est obtenue si les modes d'accès contiennent une part modérée de localité géographique. Néanmoins, cette étude suppose que l'espace de stockage est suffisamment grand pour contenir toutes les répliques. Les auteurs dans (Mansouri and Dastghaibyfard, 2012) se basent aussi sur la localité géographique. Les répliques sont stockées seulement dans certains nœuds appropriés en tenant compte de la charge de travail de chaque nœud.

D'un autre coté, quelques stratégies visent à améliorer la localité spatiale. La stratégie FIRE (*File Reunion*) proposée dans (Abdurrab and Xie, 2010) suppose qu'il existe une forte corrélation entre un groupe de tâches et un ensemble de fichiers. Elle vise à réunir un jeu de fichiers répliqué lorsqu'un des fichiers n'est pas disponible localement et qu'il y a suffisamment d'espace de stockage pour les stocker. La combinaison de la popularité des données et de la corrélation entre les fichiers a également été exploitée dans les travaux (Hamrouni et al., 2015; Qin et al., 2016). D'autres stratégies explorent des techniques de Data

Mining pour découvrir des fichiers étroitement liés (Beigrezaei et al., 2016; Hamrouni et al., 2016; Lakshmi and Thanamani, 2019). Enfin, des techniques prédictives permettent de prédire l'utilisation future des données (Khanli et al., 2011; Muthu and Kumar, 2017). Les données sont pré-répliquées sur un chemin de la source au client, de telle sorte qu'un utilisateur qui travaille dans le même contexte puisse y accéder plus rapidement à l'avenir.

2.4.1.2 Stratégies de réplication améliorant la disponibilité de la bande passante

Les stratégies de réplication citées ci-dessus tentent de maximiser la localité des données afin de réduire le temps d'accès aux données. Cependant, la capacité de stockage de chaque nœud est limitée. L'effet de la localité des données peut alors être réduit puisque seule une partie des gigantesques données produites par la grille de données peut être prise en charge par les nœuds d'une telle structure.

Certains travaux de recherche (Park et al., 2004; Horri et al., 2008; Sashi and Thanamani, 2011; Nazir et al., 2018) favorisent une autre forme de localité, appelée localité du point de vue de la bande passante du réseau. Dans ce cas, la congestion du réseau est la fonction objectif à optimiser. Ainsi, il est préférable qu'une réplique d'une entité *d* distante soit placée sur un nœud ayant une plus grande bande passante vers le nœud qui demande *d*. Dans la stratégie BHR (*Bandwidth Hierarchy based Replication*) proposée dans (Park et al., 2004), la bande passante entre les régions est inférieure à celle disponible au sein d'une région. En conséquence, les réplications intra-région sont privilégiées pour les fichiers populaires. L'algorithme 3LHA (*3-Level Hierarchical Algorithm*) proposé dans (Horri et al., 2008) prend en compte une topologie à plusieurs niveaux. Le premier niveau est constitué de régions avec une faible bande passante. Les niveaux deux et trois représentent respectivement les réseaux locaux (LAN) et les clients des réseaux locaux. Dès qu'un client accède à un fichier, celui-ci est répliqué suivant la bande passante disponible entre le nœud stockant le fichier et la région de ce client.

Les auteurs dans (Sashi and Thanamani, 2011) ont étendu la stratégie BHR. Ils supposent que les fichiers accédés par un nœud seront également accédés par les nœuds voisins et que les fichiers populaires seront consultés plus fréquemment. Les répliques ne sont placées que dans l'en-tête d'une région et sur le nœud qui effectue le plus de demandes. Enfin, les auteurs dans (Mansouri and Dastghaibyfard, 2012) ont étendu la stratégie 3LHA en proposant la stratégie DHR (*Dynamic Hierarchical Replication*). DHR gère une liste de nœuds classés par région en fonction du nombre d'accès à un fichier. Puis, une réplique est placée sur le nœud qui se trouve en haut de la liste. En plaçant les répliques sur les meilleurs nœuds, DHR vise à réduire les coûts de stockage et le temps d'exécution des travaux.

2.4.1.3 Stratégies de réplication de données visant à réduire les coûts de la réplication en se basant sur des modèles de coûts

Ces stratégies exploitent des modèles de coûts pour l'amélioration des performances du système (Lamehamedi et al., 2002; Rahman et al., 2005; Andronikou et al., 2012; Mansouri and Dastghaibyfard, 2013; Muthu and Kumar, 2017; Azari et al., 2018). La décision de réplication est généralement prise en fonction du résultat d'un modèle mathématique prenant en compte certains paramètres comme les statistiques d'accès aux fichiers, la bande passante, la latence du réseau et la taille des répliques.

Dans la stratégie proposée dans (Ranganathan et al., 2002), le nœud qui maximise la différence entre le coût total et les avantages futurs de la réplication est le meilleur client. L'utilisation d'un modèle d'estimation de coûts a également été exploitée dans la stratégie proposée dans (Lamehamedi et al., 2002). L'apport de la création d'une réplique pour l'accès aux données est évalué au moment de l'exécution, en incluant le coût de sa maintenance. Un nouveau fichier n'est répliqué que si le coût de transfert des données est réduit lors des prochaines requêtes. Enfin, la stratégie proposée dans (Zhang et al., 2010) est basée sur un modèle probabiliste afin de prédire ses performances optimales dans une grille de données hiérarchique. L'algorithme proposé réduit significativement le temps de réponse.

2.4.1.4 Stratégies de réplication de données visant à réduire les coûts de la réplication en exploitant des comportements économiques

Quelques stratégies de réplication tentent d'améliorer la QdS en exploitant des comportements économiques utilisés dans le commerce. Dans de telles stratégies, les données sont considérées comme des biens échangeables. Au cours d'une réplication, les clients ont tendance à acheter des données à des serveurs distants offrant le prix le plus bas, tandis que les nœuds distants tentent de vendre leurs données pour générer des bénéfices.

La stratégie proposée dans (Sidell et al., 1996) pour le système Mariposa (Stonebraker et al., 1996) décide de créer une réplique en se basant sur des enchères, un concept socio-économique bien connu. La meilleure réplique est sélectionnée pour une tâche donnée. Un courtier de stockage participe à ces enchères en proposant un prix auquel il vendra l'accès à une réplique si celle-ci est présente. Sinon, il lance une enchère pour répliquer le fichier demandé sur son espace de stockage s'il estime que cela est économiquement rentable. Dans la stratégie proposée dans (Carman et al., 2002), une valeur monétaire est affectée à chaque fichier. Des revenus sont générés pour un nœud lorsqu'il vend ces fichiers à d'autres nœuds. Les travaux proposés dans (Bell et al., 2003a) prévoient également les coûts et les avantages de la réplication au moyen d'un protocole d'enchères. Les auteurs dans (Cameron et al., 2004) ont également appliqué un protocole de vente aux enchères pour sélectionner une réplique parmi d'autres. Dans la stratégie proposée dans (Lin et al., 2006), un courtier de réplication est utilisé pour réduire les frais généraux de la réplication notamment en matière de transfert de données. Enfin, la stratégie proposée dans (Andronikou et al., 2012) vise à maximiser le profit tiré de la réplication. Le nombre de répliques requis est déterminé en tenant compte des contraintes infrastructurelles telles que l'équilibrage de la charge de travail sur tous les nœuds et de la bande passante disponible.

2.4.2 Classification suivant le type d'architecture de la grille de données

Il a été prouvé que les performances d'un système dépendent de la topologie sur laquelle est basé ce système. Or, les nœuds d'un système de grille de données peuvent être organisés suivant différentes topologies. En conséquence, une stratégie de réplication est conçue en fonction de la topologie pour laquelle elle a été proposée.

Dans la littérature, très peu d'études exploitent l'architecture de la grille comme un critère de classification de ces stratégies. Dans les travaux de (Grace and Manimegalai, 2014), l'architecture de la grille n'est citée que parallèlement à d'autres critères comme l'équilibrage de charge. Dans ce qui suit, nous proposons une nouvelle classification de ces stratégies suivant la topologie de grille de données pour laquelle une stratégie a été conçue (Tos et al.,

2015). Nous distinguons les stratégies conçues pour des architectures : (i) hiérarchique multi-niveaux, (ii) hiérarchique du point de vue de la bande passante, (iii) Pair à Pair (P2P), (iv) à base de graphe et (v) hybride.

2.4.2.1 Stratégies proposées pour des grilles de données avec une architecture multi-niveaux

La grille de données à plusieurs niveaux est l'exemple le plus connu de la grille de données hiérarchique. Dans une telle architecture, les nœuds sont organisés suivant une arborescence à trois niveaux ou plus. Le projet MONARC[9] a adopté une structure hiérarchique à cinq niveaux : Le niveau 0 contient toutes les données brutes générées au centre du CERN. Les données sont ensuite distribuées aux centres nationaux (par exemple, aux USA) au niveau 1. Ces derniers distribuent les données aux centres régionaux de niveau 2, qui à leur tour les distribuent aux instituts de traitement de niveau 3. Enfin, les données sont distribuées aux utilisateurs finaux au niveau 4 comme le monte la Figure 2.1 (Tos et al., 2015).

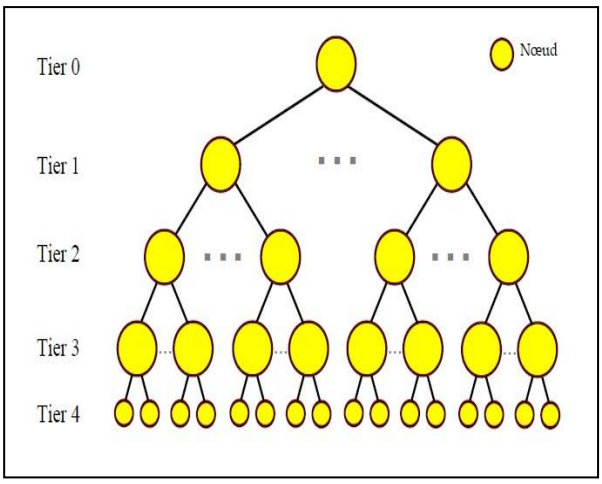

FIGURE 2.1 - Architecture hiérarchique multi-niveaux de grille de données.

Un nombre de répliques est placé à différents niveaux de la hiérarchie. Un nœud accède uniquement aux répliques stockées sur des nœuds parents ou fils (Perez et al., 2010). Puis, une demande peut monter/descendre dans l'arbre afin de rechercher la réplique la plus proche. Les stratégies SBU (*Simple Bottom-Up*) et ABU (*Aggregate Bottom-Up*) (Tang et al., 2005) exploitent la popularité des données et identifient les fichiers à répliquer en analysant l'historique d'accès aux fichiers. Lorsqu'un seuil d'accès est dépassé, SBU place les répliques à proximité des nœuds qui demandent des fichiers avec des fréquences plus élevées. D'autre part, ABU calcule les fréquences d'accès pour chaque nœud et transmet ces informations aux niveaux supérieurs jusqu'à ce que le nœud racine soit atteint. À chaque niveau, une décision de réplication est prise lorsque les valeurs d'accès dépassent un seuil prédéfini. La stratégie proposée dans (Shorfuzzaman et al., 2009) vise à trouver un compromis entre l'utilisation du stockage et le temps d'accès. Elle réplique périodiquement les fichiers en fonction de leur popularité de manière à ce que les accès soient agrégés de manière ascendante et que le placement soit effectué de manière descendante.

[9] https://monarc.web.cern.ch/MONARC/

La stratégie LALW (*Latest Access Largest Weight*) (Chang and Chang; 2008) permet de mesurer la popularité des fichiers sur la grille, de calculer le nombre requis de répliques puis, de déterminer les nœuds de placement pour ces répliques en exploitant efficacement la bande passante disponible. Les fichiers récemment consultés ont des poids plus importants et l'emplacement des répliques est basé sur les fréquences d'accès pondérées. Dans la stratégie proposée dans (Lee et al., 2012), des seuils de popularité sont utilisés afin de ne répliquer que les fichiers (20%) les plus populaires sur tous les nœuds de la grille. Cela conduit à une amélioration du temps de réponse moyen avec un coût réduit pour la réplication.

Cette topologie présente plusieurs avantages : (i) elle est facile à mettre en œuvre et (ii) permet aux nœuds d'accéder aux ressources distribuées à grande échelle de manière efficace. Généralement, la capacité de stockage augmente de bas en haut dans la hiérarchie. De plus, la structure à plusieurs niveaux permet une gestion flexible et évolutive des données et des utilisateurs. Cependant, les architectures à plusieurs niveaux ne sont pas très flexibles pour permettre l'ajout/suppression arbitraire de nœuds. À cause des règles strictes d'une structure arborescente, un seul chemin est disponible d'une feuille à la racine, c'est-à-dire qu'un nœud enfant ne peut communiquer qu'avec son parent direct. Ce type de topologie n'est alors efficace que pour les grilles de données dont l'emplacement des nœuds est connu à la conception du système.

2.4.2.2 Stratégies proposées pour des grilles de données considérant une hiérarchie du point de vue de la bande passante de réseau

Certaines institutions fournissant des ressources, via une grille de données, peuvent être géographiquement distribuées comme le montre la Figure 2.2 (Tos et al., 2015). Une telle architecture considère une hiérarchie du point de vue de la Bande Passante (BP) du réseau. Ainsi, la BP peut être assez importante au niveau d'une même institution, tandis qu'elle peut être faible entre différentes institutions.

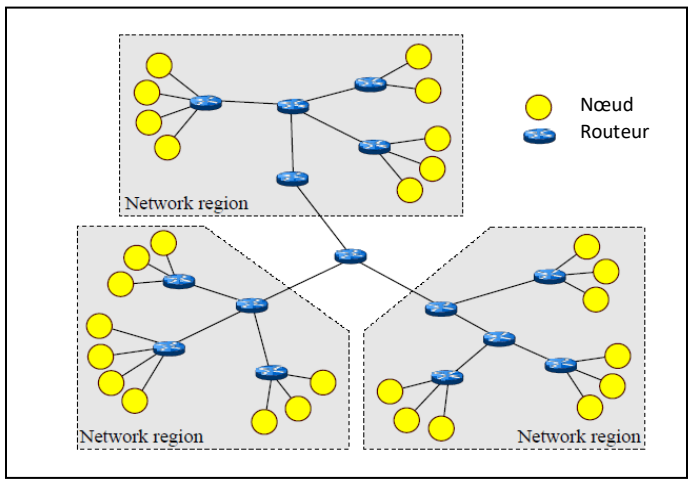

FIGURE 2.2 - Architecture de grille avec une hiérarchie du point de vue de la bande passante du réseau.

L'une des premières stratégies à exploiter la hiérarchie du point de vue de la BP est la stratégie BHR (Park et al., 2004) que nous avons décrit précédemment. Elle favorise la réplication des données populaires dans une région où la BP est abondante. De nombreuses

autres stratégies considèrent cette hiérarchie. Alors que BHR et ses variantes considèrent une hiérarchie de BP à deux niveaux, d'autres stratégies (Horri et al., 2008; Sashi and Thanamani, 2011; Mansouri et al., 2013) envisagent une hiérarchie de réseau à trois niveaux afin de représenter une topologie de réseau plus réaliste. Les auteurs dans (Mansouri et al., 2013) affirment que leur stratégie réduit sensiblement le temps de réponse par rapport à des stratégies conçues pour une hiérarchie de BP à deux niveaux.

2.4.2.3 Stratégies proposées pour des grilles de données avec une architecture P2P

Par opposition à la topologie à plusieurs niveaux, les nœuds d'une topologie P2P, appelés pairs, sont autonomes et appartiennent à un même niveau comme le montre la Figure 2.3 (Tos et al., 2015). C'est la raison pour laquelle certains travaux dans la littérature font référence à ce type de topologie par une topologie à niveau unique (Single-Tier) ou fédérée.

Les pairs s'accordent sur un ensemble commun de services et possèdent suffisamment de fonctionnalités pour agir à la fois comme serveur et comme client. De plus, ils collectent eux-mêmes les mesures des métriques utiles lors de la réplication et peuvent se connecter à n'importe quelle partie du réseau. Une certaine flexibilité dans la communication entre différents nœuds est alors possible, ce qui permet un meilleur passage à l'échelle.

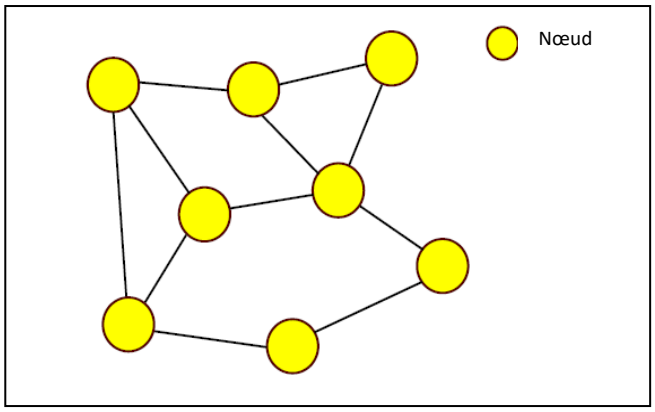

FIGURE 2.3 - Architecture de grille de données avec une topologie P2P.

De nombreuses stratégies ont été proposées pour cette topologie (Ranganathan et al., 2002; Abdullah et al., 2008; Xhafa et al., 2012; Chettaoui and Ben Charrada, 2014). La stratégie proposée dans (Ranganathan et al., 2002) permet de conserver un nombre minimum de répliques afin de satisfaire le niveau de disponibilité souhaité. La stratégie proposée dans (Challal and Bouabana-Tebibel, 2010) maximise la distance entre des répliques identiques et garantit que chaque nœud puisse disposer à proximité de répliques d'un fichier donné. D'autre part, un pair peut être membre de plusieurs groupes à la fois.

2.4.2.4 Stratégies proposées pour les grilles de données avec une architecture à base de graphe

Une topologie de graphe arbitraire est une alternative réaliste dans laquelle tout nœud peut être connecté à n'importe quel autre nœud sans aucune restriction, comme le montre la Figure 2.4 (Tos et al., 2015). C'est pour cela que ce type d'architecture cible notamment les réseaux sociaux. Les nœuds sont interconnectés avec différents niveaux de connectivité, ce

qui a une incidence sur les capacités de transfert de données vu que certains nœuds ont un accès plus facile à différentes parties de la grille. Néanmoins, l'élaboration de stratégies de réplication dans cette topologie nécessite un protocole complexe. Le fait qu'un nœud quitte ou rejoins le système engendre des coûts de maintenance élevés de l'infrastructure.

Seules quelques stratégies de réplication ont été conçues pour une grille de données avec une topologie de graphe (Lei et al., 2008; Bsoul et al., 2011; Andronikou et al., 2012; Devakirubai and Kannammal, 2013; Souravlas and Sifaleras, 2017b). La stratégie proposée dans (Chen et al., 2010) mesure le degré de distribution des nœuds dans une telle architecture et place les répliques sur les nœuds ayant des degrés plus élevés. Elle se base sur un modèle de coûts pour calculer les coûts de placement de répliques sur des nœuds candidats. Dans la stratégie EFS (Bsoul et al., 2011), le placement d'une réplique exploite le chemin le plus court entre deux nœuds dans une topologie de graphe. Enfin, la stratégie proposée dans (Andronikou et al., 2012) considère une métrique appelée 'l'importance de données' qui dépend du profit tiré de la réplication de chaque fichier. Puis, la prise de décision de réplication dépend de ce profit.

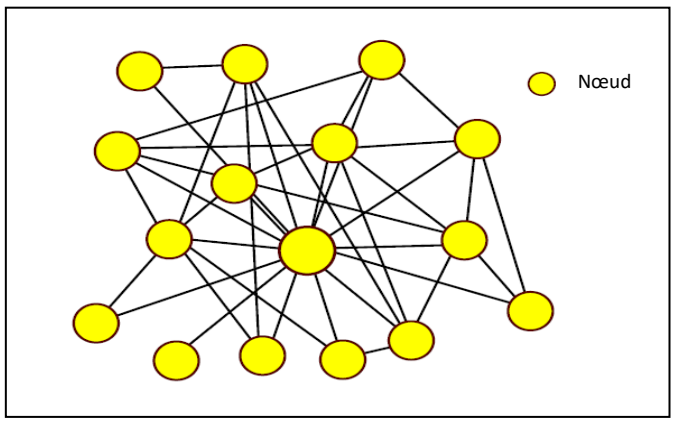

FIGURE 2.4 - Architecture de grille de données avec une topologie à base de graphe.

2.4.2.5 Stratégies proposées pour des grilles de données avec une architecture hybride

Les architectures de grille de données hybrides combinent généralement au moins deux autres architectures avec des propriétés différentes afin de profiter des avantages de plusieurs topologies. Par exemple, une combinaison entre une topologie hiérarchique et une topologie P2P suppose que les nœuds situés au même niveau d'un arbre sont connectés les uns aux autres, comme indiqué sur la Figure 2.5 (Tos et al., 2015). Cela permet d'améliorer la disponibilité et la fiabilité des données (avantages de la topologie hiérarchique) tout en garantissant un passage à l'échelle (avantage de la topologie P2P).

Seules quelques stratégies de réplication ont été proposées pour cette topologie (Lamehamedi et al., 2003; Rasool et al., 2009; Sashi and Thanamani, 2011; Azari et al., 2018). Dans la stratégie proposée dans (Lamehamedi et al., 2003), chaque nœud (sauf au niveau feuille) est connecté aux nœuds de même niveau en plus du nœud parent. Des paramètres tels que la taille d'une réplique et l'état du réseau sont récoltés en permanence afin d'estimer le coût de la réplication dont dépend la décision de réplication. La stratégie proposée dans (Azari et al., 2018) réplique les fichiers accédés par un même groupe d'utilisateurs dans une

organisation hiérarchique avec des fonctionnalités P2P entre les nœuds. La latence d'accès et la consommation en termes de BP ont été sensiblement réduites.

2.5 ANALYSE

La plupart des stratégies de réplication proposées pour les grilles de données visent à augmenter la disponibilité de données et à améliorer les performances pour des applications OLAP.

De nombreuses études dans littérature ont classé ces stratégies comme étant : (i) statiques vs. dynamiques et/ou (ii) centralisées vs. décentralisées. Nous avons proposé une nouvelle classification qui considère deux autres critères.

Le premier critère de classification correspond à la fonction objectif visée par une stratégie (Mokadem and Hameurlain, 2015). Dans un environnement aussi dynamique que la grille de données, certains objectifs sont conflictuels. Un transfert fréquent de données afin de les maintenir proches de l'utilisateur peut entraîner une surcharge des ressources du réseau. Une bonne stratégie de réplication doit alors trouver un bon équilibre entre plusieurs objectifs tels que (i) la réduction du temps d'accès, en favorisant la localité des données, (ii) la réduction de la consommation de bande passante, (iii) l'équilibrage de la charge de travail entre les nœuds et (iv) la recherche d'un nombre de répliques proche de l'optimal, ce qui nécessite un placement stratégique de ces répliques. C'est le cas de certaines stratégies de réplication (Azari et al., 2018) qui favorisent la localité de données tout en se basant sur des modèles de coûts.

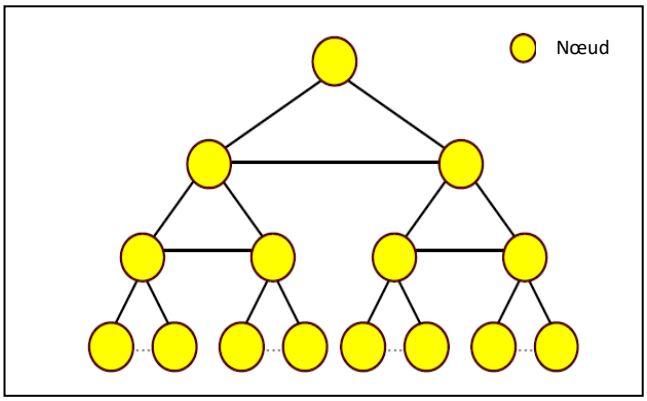

FIGURE 2.5 - Architecture hybride de grille de données.

Le second critère de notre classification correspond au type de l'architecture pour laquelle a été conçue chaque stratégie (Tos et al., 2015). En effet, une stratégie de réplication tire généralement profit de l'architecture pour laquelle elle a été conçue. Dans ce contexte, le placement de répliques sur les différents niveaux d'une architecture hiérarchique permet d'améliorer l'accès aux données. C'est pour cela que la plupart des stratégies dans la littérature ont été proposées pour ce type d'architecture, facile à mettre en œuvre. Toutefois, l'emplacement d'une réplique doit être soigneusement pris en compte.

					Classification basée sur le type de l'architecture	Classification basée sur la fonction objectif			
	Simulation	Hypothèse/ Ressource Stockage	Scalabilité	Prise en compte des coûts	Topologie de la grille	Localité de données	Localité bande passante	Comportement économique	Modèle de coûts
Ranganathan and Foster, 2001	X	Illimité	-	-	Multi-niveaux	X	-		-
Ranganathan et al., 2002	X	Illimité	-	X	P2P	X	-		-
Carman et al., 2002	X	Illimité	-	X	Multi-niveaux	-	-	X	-
Bell et al., 2003a	X	Illimité	X	X	Multi-niveaux	-	-	X	-
Lamehamedi et al., 2003	X	Limité	X	X	Hybride	-	-	-	X
Park et al., 2004	X	Limité	-	X	Hiérarchique BP	-	X	-	-
Cameron et al., 2004	X	Limité	-	X	Multi-niveaux	-	-	X	-
Tang et al., 2005	X	Limité	-	X	Multi-niveaux	X	-	-	-
Chang and Chang; 2008	-	Limité	-	X	Multi-niveaux	X	-	-	-
Rahman et al., 2008	X	Lllimité	X	-	Graphe	-	-	-	X
Rasool et al., 2009	X	Limité	-	-	Hybride	X	-	-	-
Zhang et al., 2010	-	Illimité	-	X	Multi-niveaux	-	-	-	X
Bsoul et al., 2011	X	Limité	-	X	Graphe	X	-	-	-
Sashi and Thanam., 2011	X	Limité	-	X	Hybride	-	X	-	-
Nukarapu et al., 2011	X	Limité	-	X	Multi-niveaux	X	-	-	-
Lee et al., 2012	X	Illimité	-	X	Multi-niveaux	X	-	-	-
Andronikou et al., 2012	X	Limité	X	X	Graphe	X	-	X	-
Mansouri et al., 2013	X	Limité	X	X	Hiérarchique BP	-	X	-	-
Chettaoui and Ben Cherrada, 2014	X	Limité	X	-	P2P	X	-	-	X
Muthu and Kumar, 2017	X	Limité	X	X	Multi-niveaux	X	-	-	X
Omer et Abdella, 2018	X	Limité	-	X	Multi-niveaux	-	X	-	-

TABLE 2.1 - Caractéristiques et classification de certaines stratégies de réplication dans les grilles de données.

Bien qu'il n'y ait pas de point de défaillance unique dans une topologie P2P, une décision décentralisée peut conduire à la création inutile de répliques vu qu'un pair peut avoir une

vision partielle du système. Cela motive l'introduction d'une architecture hybride qui exploite les avantages de plusieurs architectures afin d'aboutir à de meilleures performances. D'un autre coté, la plupart des travaux ont mentionnés l'extension de leurs stratégies aux grilles de données avec une topologie de graphe. Une telle topologie est plus réaliste. Cependant, seules certaines stratégies ont été proposées pour ce type d'architecture (Andronikou et al., 2012; Souravlas and Sifaleras, 2017b). Les arrivées/départs fréquents de nœuds engendrent un coût de maintenance élevé, ce qui dégrade les performances. Pour cela, certaines stratégies (Rahman et al., 2008) proposent des algorithmes de maintenance pour déplacer les répliques vers d'autres nœuds lorsqu'un nœud quitte le système. Afin de mieux comparer ces stratégies, nous avons identifié dans la Table 2.1, une liste non exhaustive de quelques stratégies en se positionnant par rapport à la classification que nous avons proposé dans les grilles de données.

Nous avons également évalué l'impact de plusieurs facteurs sur les performances d'une stratégie de réplication en utilisant le simulateur OptorSim (Bell et al., 2003b). Dans ce contexte, on ne retrouve pas beaucoup d'études comparatives entre les stratégies proposées dans la littérature. Dans la plupart des cas, les stratégies proposées ont été validées au travers de comparaison avec les résultats de stratégies de base telles que LFU (*the Last Frequently Used*) et LRU (*the Least Recently Used*) (Arlitt et al., 2000). Cela est dû au fait que ces algorithmes de remplacement de répliques sont déjà implémentés dans OptorSim.

Nos expériences ont montré que la consommation de la bande passante et la capacité de stockage sont des facteurs très importants qui ont une incidence sur les performances d'une stratégie de réplication (Mokadem and Hameurlain, 2015). La prise en compte des modes d'accès aux données est également essentielle pour une évaluation réaliste des performances d'une stratégie de réplication. Or, cela est ignoré dans la plupart des stratégies existantes dans la littérature. Un fichier récemment ajouté à la grille peut gagner en popularité puis, peut être moins accédé par la suite. Dans ce contexte, seules quelques stratégies ont été évaluées sous différents modes d'accès, par exemple 3 modes dans (Ranganathan and Foster, 2001) et plus de 5 modes d'accès dans (Lee et al., 2012).

2.6 CONCLUSION

Nous avons discuté des problèmes liés à la réplication de données dans les systèmes de grille de données. L'étude de l'état de l'art nous a permis de proposer une nouvelle classification basée sur des critères tels que la fonction objectif visée par une stratégie et le type d'architecture de grille pour laquelle une stratégie a été conçue. Une bonne stratégie de réplication de données doit trouver un bon équilibre entre plusieurs fonctions objectif. Elle doit également tirer profit de la topologie de la grille de données pour laquelle elle a été conçue.

Les grilles de données sont construites à des fins spécifiques et leurs ressources sont partagées entre différentes institutions de manière fédérée (Foster et al., 2008). Maximiser l'utilisation de ces ressources pour maximiser les avantages souhaités est un objectif commun à ces stratégies. Certaines stratégies font aussi l'hypothèse d'abondance de ressources (Ranganathan et al., 2002) quand d'autres stratégies visent à augmenter la disponibilité en exploitant au maximum les ressources de stockage disponibles. Cela n'est pas très réaliste et a forcément des répercussions non négligeables. Un fardeau économique est alors généré pour les institutions qui fournissent ces ressources. Bien que le coût de stockage devienne

raisonnablement bas de nos jours, toute étude expérimentale d'une stratégie de réplication doit considérer des ressources finies. En d'autres termes, bien que les ressources d'une grille de données soient réparties entre des institutions de manière fédérée, elles ne sont pas toujours gratuites.

Chapitre 3

Réplication de données dans les systèmes Cloud

Résumé

Dans ce chapitre, nous commençons par mettre en évidence les caractéristiques clés des environnements Cloud. Ensuite, nous présentons les principales stratégies de réplication de données dans les systèmes Cloud à travers les classifications existantes de ces stratégies dans la littérature. Enfin, nous proposons une nouvelle classification de ces stratégies dans les systèmes Cloud.

Sommaire

3.1 INTRODUCTION

Durant la dernière décennie, le Cloud Computing (l'informatique en nuage) s'est imposé comme un paradigme informatique populaire (Buyya et al., 2009). Dans un environnement Cloud, les traitements peuvent concerner de gigantesques volumes de données distribuées à travers des Centres de Données (CDs), eux même distribués à grande échelle. Cette explosion du volume de données résulte, d'une part, du fait qu'il est plus facile de collecter

les données (par exemple, via des fichiers Log ou des capteurs) et, d'autre part, des coûts de plus en plus bas des périphériques de stockage (Hameurlain and Morvan, 2016). Des milliers d'utilisateurs, appelés locataires, accèdent fréquemment à ces données avec l'apparence de disposer de ressources illimitées, par exemple des ressources de stockage et/ou de calcul (Foster et al., 2008).

D'un coté, assurer une certaine QdS pour les locataires tout en mutualisant ces ressources entre plusieurs locataires est essentiel pour les fournisseurs de Cloud. Dans ce contexte, la réplication de données permet de répondre à une certaine QdS pour les locataires, par exemple en termes de disponibilité de données, de réduction des temps d'accès ou de tolérance aux pannes. D'un autre coté, les fournisseurs de Cloud aspirent à générer un certain profit économique (Hameurlain and Mokadem, 2017). Pour cela, ils se basent sur une gestion élastique des ressources allouées aux locataires. Ces derniers payent l'utilisation de ces ressources suivant le modèle 'pay as you go' (Armbrust et al., 2010), c'est-à-dire qu'ils ne payent au fournisseur que ce qu'ils consomment comme ressources. Ces caractéristiques, en plus d'autres raisons que nous évoquons plus loin, font que les stratégies de réplication de données proposées initialement dans les systèmes de grille ne sont pas adaptées aux systèmes Cloud.

Dans ce qui suit, nous mettons en évidence les concepts de base du Cloud Computing. Ensuite, nous évoquons les problèmes liés à la réplication de données dans les systèmes Cloud à grande échelle. Nous passons en revue un large état de l'art à travers les classifications existantes des stratégies de réplication proposées dans la littérature. Enfin, nous proposons une nouvelle classification de ces stratégies dans de tels systèmes.

3.2 CLOUD COMPUTING : CONCEPTS DE BASE

3.2.1 Contexte et définitions

Le Cloud Computing consiste à proposer des services informatiques à la demande à des locataires. Les auteurs dans (Foster et al., 2008) l'ont défini comme suit :

> *A large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualized, dynamically-scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers over the Internet.*

Les auteurs dans (Vaquero et al., 2009) ont définis le Cloud comme un vaste regroupement omniprésent de ressources informatiques facilement accessibles et qui peuvent être dynamiquement reconfigurées pour s'adapter à une charge variable, ce qui permet une utilisation optimale des ressources. De nombreuses autres définitions existent dans la littérature. C'est pour cela que l'Institut National des Standards et technologies (INST) a distingué les principales caractéristiques du Cloud comme suit (Mell et al., 2011) :

(i) Les ressources sont approvisionnées (sous forme de services) aux locataires à la demande, sans interruption, en libre service et sans interaction humaine avec le fournisseur de services.

(ii) Les services sont accessibles via le réseau.

(iii) Les ressources sont mutualisées entre plusieurs utilisateurs qui ne peuvent investir dans des infrastructures avec autant de ressources.

(iv) Une élasticité rapide consiste à provisionner et libérer automatiquement des ressources suivant le besoin de chaque application avec une impression de ressources illimitées pour l'utilisateur.

(v) Ces ressources sont contrôlées et optimisées par le système de manière transparente aussi bien pour le fournisseur que pour les locataires.

(vi) Un service doit être mesurable et les locataires paient le loyer correspondant aux services acquis au fournisseur, selon le modèle de tarification 'pay as you go'.

(vii) Bien qu'avec du matériel standard, un système Cloud doit être tolérant aux pannes matérielles et logicielles à travers des mécanismes telles que la réplication.

3.2.2 Caractéristiques du Cloud

Le fournisseur du Cloud vise à maximiser ses bénéfices tandis que l'objectif d'un locataire est d'obtenir une QdS acceptable à un faible coût. La prise en compte des concepts économiques est alors une caractéristique importante dans le Cloud Computing. Le bénéfice d'un fournisseur correspond à la différence entre ses revenus (ce que payent les locataires pour les services) et ses dépenses liées à l'exécution des requêtes des locataires. Une section est consacrée au modèle économique dans le Cloud (incluant le profit du fournisseur) dans le chapitre suivant.

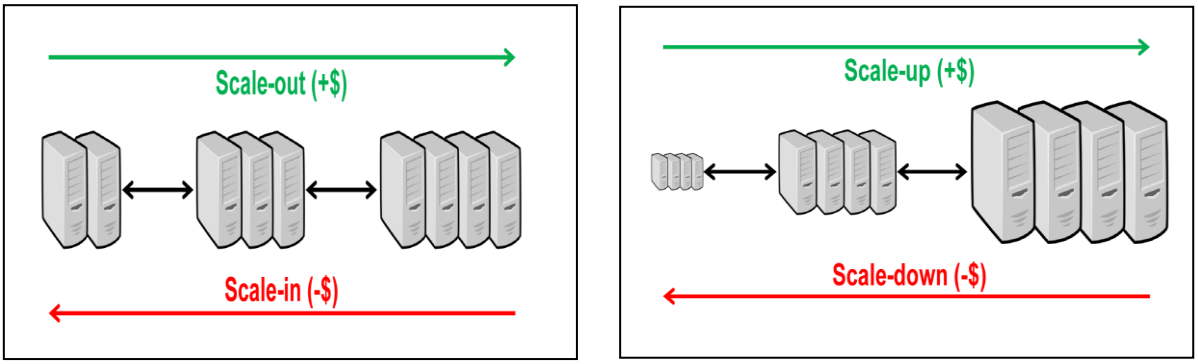

FIGURE 3.1 - Illustration de l'élasticité horizontale (à gauche) et de l'élasticité verticale (à droite).

La relation entre le fournisseur et ses locataires est établie au moyen d'un contrat de niveau de service SLA (*Service Level Agreement*) (Stantchev and Schröpfer, 2009; Buyya et al., 2009). C'est un contrat juridiquement contraignant qui réglemente et protège les intérêts des parties impliquées. Il a été introduit initialement pour gérer la QdS dans le domaine des télécommunications. Un SLA comprend principalement (Kouki and Ledoux, 2012) :

(i) Un ou plusieurs objectifs (exigences) de niveau de service SLO (*Service Level Objective*) pour le locataire, à satisfaire par le fournisseur, par exemple une disponibilité de données et des performances en termes de temps de réponse.

(ii) Les parties concernées par ce contrat.

(iii) Une période de validité du SLA ainsi qu'une Période de Facturation (PF) correspondant à la durée pendant laquelle le locataire loue des ressources qu'il paye au fournisseur.

(iv) Un montant convenu payé par le locataire au fournisseur pour l'utilisation des ressources (traitement des requêtes et stockage) au cours de la PF.

(v) Un montant convenu d'une amende payée par le fournisseur à son locataire en cas de non respect du SLA. En effet, des garanties sont offertes par le fournisseur à chaque locataire pour l'utilisation des ressources. La violation des conditions du SLA entraîne souvent des pénalités modélisées sous forme de compensations monétaires payées par le fournisseur au locataire concerné (Xiong et al., 2011).

En plus du contrat SLA, d'autres concepts tels que la *virtualisation* (Smith and Nair, 2005) et *l'élasticité* (Al-Dhuraibi et al., 2017) sont aussi importants dans les systèmes Cloud :

(i) La virtualisation (qui n'est pas spécifique aux Clouds) permet d'exécuter plusieurs applications et de créer des Machines Virtuelles (MVs) sur une machine physique. Elle apporte de nombreux avantages tels qu'une utilisation optimale des ressources existantes et l'économie sur le matériel par mutualisation. Cela permet la perception qu'une ou plusieurs entités existent, bien que les entités ne soient pas physiquement présentes. Des ressources de calcul et un volume de stockage sont alloués à chaque MV.

(ii) L'élasticité correspond à la capacité d'un système à s'adapter dynamiquement (en approvisionnant/désapprovisionnant des ressources) et de manière automatique à la charge de travail tout en maintenant une QdS pour les locataires. Le dimensionnement automatique correspond au mécanisme permettant de décider à quel moment rajouter ou diminuer des ressources en fonction de la charge de travail. Il existe deux types d'élasticité : (a) L'élasticité horizontale correspond à la possibilité d'augmenter (Scaling-out) ou diminuer (Scaling-in) le nombre de MVs du système. D'un autre coté, (b) l'élasticité verticale correspond à la possibilité d'augmenter (Scaling-up) ou de diminuer (Scaling-down) le nombre de ressources (CPU, mémoire,…) dans une machine physique (Graefe et al., 2013) comme le montre la Figure 3.1. L'avantage de l'élasticité verticale est l'amélioration sensible des performances. Néanmoins, cette solution est chère et souvent contrainte par des limites matérielles. D'ailleurs, les auteurs dans (Hwang et al., 2016) ont démontré que le 'Scaling-out' entraîne moins de provisionnement de ressources et offre une meilleure extension du système que le 'Scaling-up'. Concernant la prise de décision, il existe de nombreuses approches de dimensionnement automatique. On citera notamment : (a) le dimensionnement basé sur les seuils, qui consiste à ajouter/supprimer des ressources lorsque la valeur observée (Han et al., 2012) ou estimée (Khatua et al., 2010; Bai et al., 2013; Tos et al., 2016) d'une certaine métrique dépasse ou est au dessous d'un seuil prédéfini, et (b) le dimensionnement basé sur l'apprentissage par renforcement, qui permet d'éviter l'intervention humaine au prix d'un temps d'apprentissage pour un agent qui apprend de ses actions successives (Rao et al., 2011).

Dans nos travaux, nous avons adopté une élasticité horizontale à travers le rajout/suppression de répliques dans le système. Concernant le dimensionnement automatique des ressources, nous considérons l'approche basée sur les seuils lors de la prise de décision de réplication. Par ailleurs, l'allocation de telles ressources est faite de telle sorte qu'un fournisseur et ses locataires s'accordent toujours sur le contrat SLA (Da Silva et al., 2012). Une solution élastique (Figure 3.2 - c) permet alors de remédier au sur-approvisionnement et au sous-approvisionnement des ressources :

(i) Un sur-approvisionnement de ressources se produit lorsque les ressources allouées par le système dépassent les ressources nécessaires (Figure 3.2 - a). Même si la QdS est satisfaite, cela entraine des coûts supplémentaires et inutiles pur le fournisseur, pouvant

être vu comme un gaspillage de ressources. Ceci peut provoquer une indisponibilité de ressources pour les autres locataires.

(ii) Un sous-approvisionnement des ressources se produit lorsque les ressources allouées par le système sont inférieures aux ressources requises (Figure 3.2 - b). Ceci peut provoquer une dégradation des performances et le payement de pénalités par le fournisseur aux locataires concernés. Il peut également conduire au mécontentement des locataires concernés qui pourraient résilier leur contrat avec le fournisseur.

Les auteurs dans (Al-Dhuraibi et al., 2017) revisitent les travaux portant sur l'élasticité. Ainsi, les solutions d'élasticité sont classées en fonction de la configuration (rigide, configurable), de la portée (infrastructure, plateforme), de l'objectif (performances, coûts, énergie, disponibilité), du mode (réactif, prédictif), de la méthode (horizontale, verticale, hybride), de l'architecture (centralisée, décentralisée) et du fournisseur (unique, multiple).

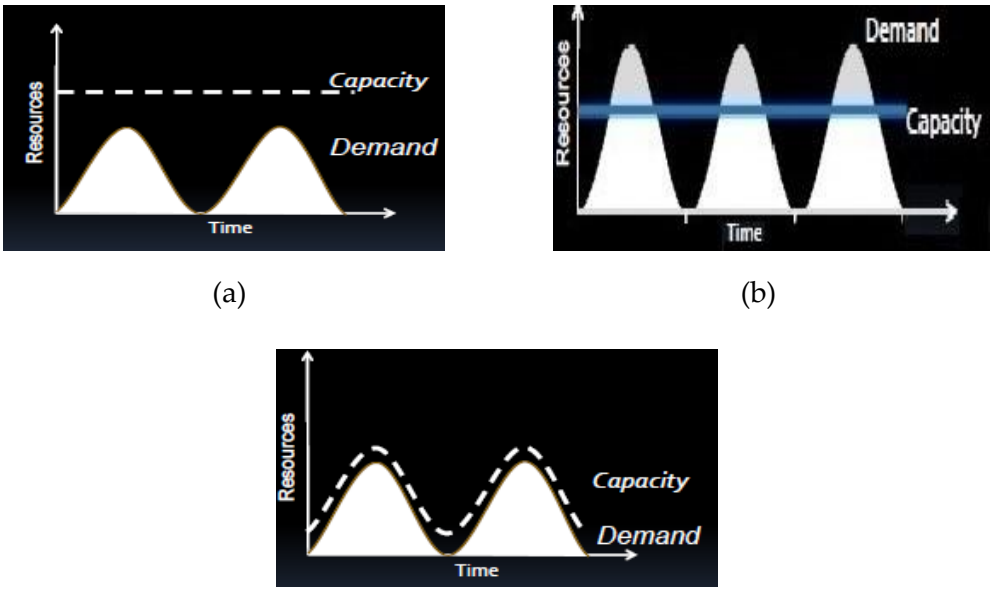

FIGURE 3.2 - (a) Sous-approvisionnement, (b) sur-approvisionnement et (c) ajustement automatique des ressources.

3.2.3 Modèles de services et domaines d'application

Le Cloud Computing couplé aux technologies déjà existantes auparavant telle que la virtualisation permet la mise à disposition d'infrastructures, de plateformes et d'applications à la demande (Buyya et al, 2009). Comme le montre la Figure 3.3 (Kouki, 2013), les principaux modèles de services proposés par un fournisseur Cloud sont :

(i) L'infrastructure en tant que service IaaS (*Infrastructure as a Service*). Dans un tel service (Manvi and Shyam, 2014), le fournisseur loue des capacités de calcul fournies par des infrastructures telles qu'Amazon EC2 (*Amazon Elastic Compute Cloud*), des ressources de stockage, par exemple à travers Amazon S3, ainsi qu'une connectivité réseau aux locataires. La souscription à une offre IaaS permet à un locataire d'être dispensé de

l'achat de matériel informatique, d'externaliser son parc matériel, et de s'affranchir des compétences de conception et d'exploitation des infrastructures techniques.

(ii) La plateforme en tant que service PaaS (*Platform as a Service*) (Pokahr and Braubach, 2016). Dans un tel service, par exemple *Microsoft Azure* ou *Google App Engine*, le locataire loue l'exploitation de serveurs sur lesquels les outils nécessaires sont préalablement placés et contrôlés par le fournisseur comme, par exemple le système d'exploitation ou le SGBD. Un fournisseur Paas peut être vu comme un locataire faisant appel à un fournisseur IaaS pour la gestion du réseau, des machines physiques et de la virtualisation. Le locataire peut avoir le contrôle des applications. Le PaaS est destiné principalement aux développeurs d'applications et administrateurs de bases de données.

(iii) Le logiciel en tant que service SaaS (*Software as a Service*) (Dutt et al., 2018). Dans un tel service, par exemple les applications *App de Google* ou *Exchange on Line* de Microsoft, le locataire loue les applications mises à disposition par le fournisseur. Le locataire externalise ses applications auxquelles il accède à la demande. Il paie à l'usage, selon le nombre d'utilisateurs et/ou le temps d'utilisation du logiciel. Ces applications peuvent être manipulées à l'aide d'un navigateur Web ou installées de façon locative sur un PC. Leurs administrations, configuration et mises à jour sont assurées par le fournisseur qui est responsable du matériel, du système d'exploitation et de la virtualisation.

Par ailleurs, de nombreux domaines d'application s'appuient sur des infrastructures de Cloud. On citera les domaines : militaire (Joint Enterprise Defense Infrastructure, 2019), de santé (MedicalCloud, 2020), de l'éducation (AWS Educate, 2020) et de l'industrie (Salesforce, 2020).

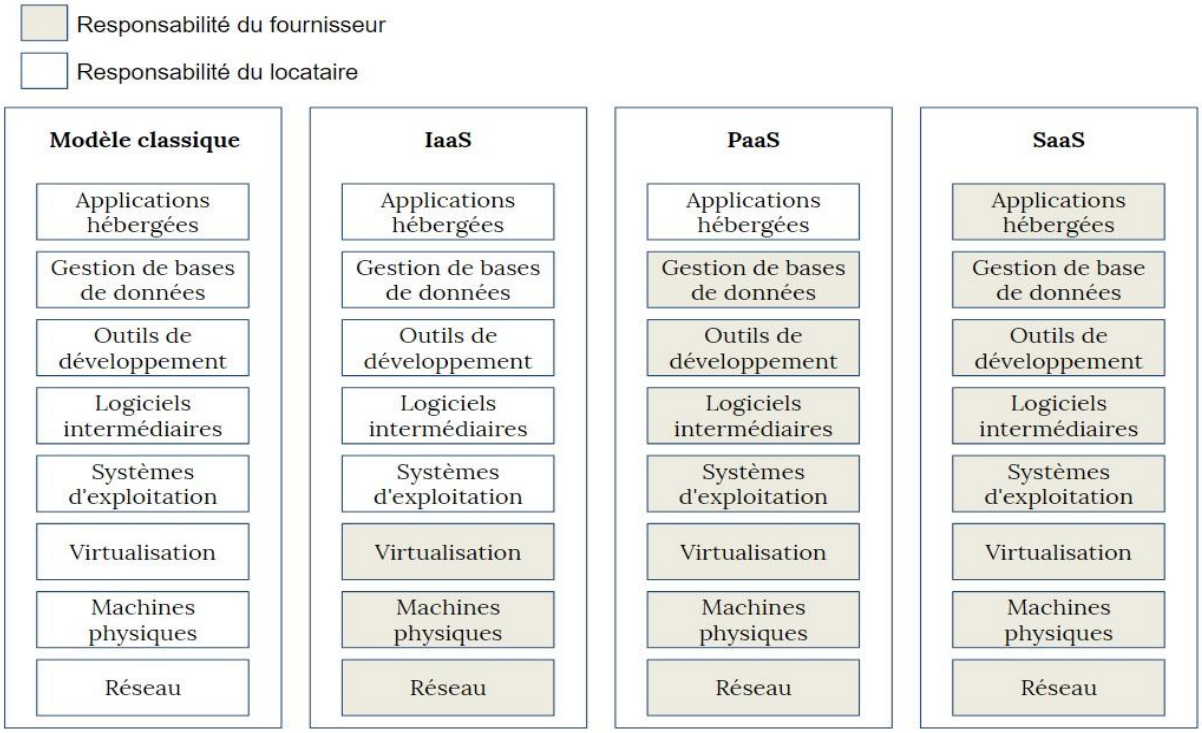

FIGURE 3.3 - Modèles de services du Cloud.

3.2.4 Déploiement du Cloud

L'utilisation d'un Cloud permet à une entreprise de réduire ces coûts en termes d'infrastructures soit en achetant un serveur composé de machines de base à bas prix, soit en louant des services (IaaS, PaaS, SaaS) à un fournisseur (Hameurlain and Morvan, 2016). Il existe trois modèles de déploiement du Cloud : public, privé ou hybride :

(i) Le Cloud public (Persico et al., 2018) représente le Cloud traditionnel ouvert au grand public. Le fournisseur procède à une mutualisation des ressources avec un nombre, en théorie, illimité de clients. Une telle offre privilégie l'élasticité et la flexibilité avec des prix plus attractifs pour ces clients. N'importe quel particulier peut y héberger ses applications, services ou données. Pour les consommateurs, il n'y a donc aucun investissement initial et aucune limite de capacité. Malgré ces avantages, ce type de déploiement présente certains inconvénients relatifs à la confidentialité des données en plus du risque de perte de données.

(ii) Dans le Cloud privé (Chang, 2015), les ressources physiques sont hébergées dans une infrastructure propre à une seule organisation et étant sous son contrôle. En conséquence, c'est à sa charge de contrôler le déploiement des applications. Il peut aussi désigner un Cloud déployé sur une infrastructure physique externe. Des ressources dédiées sont alors fournies par un fournisseur qui répond aux exigences des clients souhaitant des garanties exclusives sur un contrat de service. Par rapport au Cloud public, le Cloud privé permet une meilleure sécurité et une meilleure gestion de coûts.

(iii) Un Cloud hybride (Azumah et al., 2019) est l'utilisation de plusieurs infrastructures Cloud, publics ou privés. Il s'agit de faire cohabiter ces deux environnements afin de profiter de leurs avantages. L'environnement privé est réservé aux systèmes courants, tandis que les capacités du Cloud public sont utilisées pour absorber ponctuellement les montées en charge pour permettre un meilleur passage à l'échelle.

3.2.5 Bases de Données (BD) dans le Cloud

De nombreux Cloud commerciaux fournissent des solutions pour la gestion des BDs. Cela permet aux entreprises d'éviter un grand investissement matériel, humain et technique. Certains fournisseurs proposent la location d'instances de MVs puissantes et optimisés que le locataire peut utiliser librement en installant son SGBD et en s'occupant de sa maintenance (Abadi, 2009). Oracle Virtual Box, Amazon Elastic Compute Cloud (EC2) ou Azure Cloud Shell sont des exemples de plateformes offrant cette solution qui convient mieux à une utilisation limitée dans le temps.

L'autre solution, largement proposée par les Cloud commerciaux et que nous considérons dans nos travaux, consiste à fournir des services de gestion de BD (Agrawal et al., 2009) considérés comme des produits de niveau PaaS. Ces services sont utilisés soit par les utilisateurs finaux soit par des services de niveau supérieurs (SaaS) et font appel aux services du niveau IaaS qui fournissent des capacités de calcul et de stockage. Dans cette solution, le fournisseur est responsable de la maintenance de la BD et facture l'utilisation d'un tel service en prenant en compte des paramètres tels que la durée de l'utilisation du service. Certains travaux font référence à ces solutions par l'offre de BD en tant que service DBaaS (*Database as a Service*). Cela correspond à la mise à disposition d'un accès à une BD via un SGBD pour des

locataires sans qu'il soit nécessaire de configurer un matériel physique ou d'installer un logiciel quelconque.

Plusieurs approches sont possibles pour stocker et traiter des volumes de données souvent volumineux. On notera l'utilisation des : (i) systèmes NoSQL (*Not Only SQL*), tels que les services DynamoDB d'Amazon[10], DocumentDB de Microsoft[11] et Cloud Bigtable de Google[12], (ii) les SGBD relationnel tels que les services cités précédemment (Relational Database Service (RDS) d'Amazon, Azure SQL Database de Microsoft, Cloud SQL de Google et Oracle Database Cloud service), ou encore (iii) l'environnement Hadoop, par exemple les services Elastic MapReduce d'Amazon[13], HDInsight de Microsoft[14], Cloud Dataproc de Google[15] et Big Data Cloud Service d'Oracle[16]. Dans ce contexte, une comparaison intéressante des services de BD des principaux fournisseurs Cloud du marché est fournie dans (Kandi, 2019). Elle concerne notamment le type des données considérées (relationnel, NoSQL), l'infrastructure de calcul, les paramètres de tarification (capacité du matériel, nombre de nœuds, nombre de BD), la nature de l'élasticité (horizontale ou verticale) et les objectifs mentionnés dans le SLA.

Dans nos travaux de recherche, nous nous intéressons principalement, à la mise en commun des ressources en considérant un fournisseur PaaS public. Le but est de tirer profit des ressources du Cloud en termes de puissance de calcul et de stockage de données lors de l'exécution des requêtes relationnelles des locataires. Dans ce qui suit, nous nous intéressons aux problèmes introduits par la réplication de données dans les systèmes Cloud.

3.3 PROBLEMES LIES A LA REPLICATION DE DONNEES DANS LES SYSTEMES CLOUD

Les stratégies de réplication de données proposées pour les systèmes de grille de données ne sont pas adaptées aux systèmes Cloud. Elles visent à obtenir les meilleures performances sans tenir compte des concepts économiques tels que le coût économique de la réplication ou le profit des fournisseurs Cloud. La création d'autant de répliques dans les systèmes Cloud n'est pas économiquement faisable car cela peut entrainer une consommation inutile des ressources et donc, une réduction des bénéfices du fournisseur. En conséquence, les stratégies de réplication dans les systèmes Cloud doivent assurer une certaine QdS pour les locataires tout en tenant compte de la rentabilité économique pour le fournisseur.

Une stratégie de réplication de données dans les systèmes Cloud doit alors non seulement répondre aux problématiques classiques citées dans le chapitre précédent, c'est-à-dire : (i) *Quand* la réplication doit-elle avoir lieu ? (ii) *Quelles* données doivent être répliquées ? (iii) *Où* placer ces répliques ? et (iv) *Combien* de répliques doivent être créées ? Mais aussi, elle doit prendre en compte les *coûts économiques* liés à la réplication. En d'autres termes, il faut également répondre à la problématique : *Combien coûte cette réplication et est-elle profitable* ?

[10] https://aws.amazon.com/fr/dynamodb/
[11] https://sauget-ch.fr/category/azure/documentdb/
[12] https://cloud.google.com/bigtable/?hl=fr
[13] https://aws.amazon.com/fr/emr/
[14] https://azure.microsoft.com/fr-fr/services/hdinsight/
[15] https://cloud.google.com/dataproc/?hl=fr
[16] https://www.oracle.com/fr/big-data/

Contrairement aux systèmes de grille de données, un système Cloud ne vise pas la meilleure QdS possible pour le locataire. Le rapport qualité/prix devient alors très important pour celui-ci et doit faire, au préalable, l'objet de négociations entre le fournisseur et chaque locataire au moment de signer le contrat SLA (Yin et al., 2018). D'un autre coté, un fournisseur de Cloud vise à générer un profit économique, ce qui nécessite une gestion élastique des ressources. Ceci permet d'augmenter ou de diminuer les ressources allouées à un locataire à la demande suivant la charge de travail. Par exemple, le nombre de répliques peut être augmenté dans le but de maintenir une certaine QdS si les termes du SLA ne sont pas respectés, ce qui évite au fournisseur le payement de pénalités à ses locataires.

Dans la littérature, un grand nombre de stratégies de réplication de données dans les systèmes Cloud visent à garantir une certaine QdS pour les locataires au travers de la satisfaction de certains objectifs SLOs figurant dans le SLA et que le fournisseur doit absolument respecter. On citera notamment l'objectif de disponibilité de données (Wei et al., 2010; Silvestre et al., 2012; Sun et al., 2012a; Long et al., 2014; Gill and Singh, 2016; Mansouri et al., 2017; Edwin et al., 2019) ou de tolérance aux fautes (Qu and Xiong, 2012; Vijayakumar et al., 2015; Li et al., 2019). La plupart de ces stratégies n'intègrent pas l'objectif de temps de réponse dans le SLA. Il en est de même pour la plupart des fournisseurs de Cloud commerciaux (Amazon, Google et Microsoft) qui n'offrent pas, en général, de garanties de temps de réponse, en tant que partie intégrante du SLA. Par exemple, Google Cloud SQL[17] ne fournit que des garanties contre les temps d'arrêt et les erreurs, sans garantie en termes de temps de réponse. Cela s'explique par les charges de travail hétérogènes dans les systèmes Cloud. En effet, la satisfaction de performances pour le locataire et la garantie d'un bénéfice économique maximum, avec des coûts d'exploitation minimum, constituent des objectifs contradictoires (Long et al., 2014). Cependant, les stratégies de réplication de données dans de tels systèmes doivent prendre en compte le compromis entre les performances pour les locataires et le profit économique pour le fournisseur, en particulier lorsque ces stratégies sont proposées pour des applications OLAP, comme c'est le cas dans nos travaux.

3.4 CLASSIFICATION DES STRATEGIES DE REPLICATION DE DONNEES DANS LES SYSTEMES CLOUD

Un certains nombre de travaux de synthèse (Gill and Singh, 2016; Milani and Navimipour, 2016; Gopinath and Sherly, 2018a) se sont intéressés à la classification des principales stratégies de réplication de données proposées dans les systèmes Cloud. La plupart de ces classifications sont basées sur des critères tels que la nature de la stratégie et/ou le mécanisme de contrôle de la réplication, permettant ainsi de classer ces stratégies comme étant : (i) statiques vs. dynamiques et/ou (ii) centralisées vs. décentralisées respectivement.

Stratégies statiques vs dynamiques. Dans une stratégie de réplication statique (Ghemawat et al., 2003; Hassan et al., 2009; Kirubakaran et al., 2013; Long et al., 2014; Zeng and Verravalli, 2014; Begum and Sirisha, 2019), le nombre de répliques pour une unité de données est déterminé à l'avance pendant la phase de conception. Des politiques déterministes sont appliquées afin de décider, à l'avance, de l'emplacement de chaque réplique.

[17] https://cloud.google.com/sql/

Les auteurs dans (Ghemawat et al., 2003) ont proposé un algorithme de réplication de données dans GFS (*Google File System*) offrant un temps de réponse réduit et une haute disponibilité. Un nombre fixe (3) de répliques est utilisé pour tous les fichiers. Ceci n'est pas forcément avantageux vu la consommation importante des ressources notamment en termes de stockage et d'énergie. La stratégie de réplication proposée dans (Hassan et al., 2009) permet la réduction du stockage et de la latence ainsi que l'amélioration de la fiabilité des données dans un cluster de stockage à grande échelle. Cette solution explore un espace de recherche dans lequel une solution potentiellement bonne peut être trouvée. MORM (Long et al., 2014) est une autre stratégie de réplication statique qui se base sur un algorithme de réplication hors ligne. Elle vise à améliorer la disponibilité de données, le temps de réponse et la latence du réseau. Un schéma de réplication est établi à l'avance pour une longue période en fonction de la capacité initiale du système. Dans la stratégie proposée dans (Zeng and Verravalli, 2014), une technique d'équilibrage de charge est considérée pour les CDs composés de milliers de serveurs de données brutes et de centaines de serveurs de métadonnées connectés par des réseaux arbitraires. Cette technique gère plutôt la réplication des métadonnées. Elle vise à minimiser le temps de réponse.

Dans une stratégie de réplication dynamique (Bui et al., 2016, Qu et al., 2016 ; Tos et al., 2018; Mansouri and Javidi, 2018a; Gopinath and Sherly, 2018b; Tabet et al., 2019; Mokadem and Hameurlain, 2020), les répliques d'une unité de données sont créées, placées et gérées de manière dynamique lorsque le système est déjà opérationnel. Cette création intervient en fonction des modes d'accès de l'utilisateur et de la disponibilité des ressources, par exemple de stockage et de bande passante (Li et al., 2011; Lei et al., 2008). Les principales phases dans ce type de stratégies sont : l'analyse de la relation entre le nombre de répliques et la disponibilité des ressources, l'identification des données populaires, l'évaluation de la bande passante du réseau et l'équilibrage de charge entre les nœuds pouvant recevoir des répliques.

Dans la stratégie proposée dans (Wei et al., 2010), une réplique est placée sur le nœud ayant la plus petite probabilité de blocage. Celle-ci est calculée sur chaque MV afin de mesurer sa charge de travail. Puis, les MVs surchargées sont bloqués pour la réception de nouvelles requêtes des locataires. La stratégie proposée dans (Sousa and Machado, 2012) prend en compte les performances d'un SGBD. Lorsque la charge de travail augmente, elle crée des nouvelles répliques ou redirige les requêtes vers les répliques disposant d'assez de ressources. La stratégie proposée dans (Bai et al., 2013) se base sur un seuil de temps de réponse. Elle crée de nouvelles répliques lorsque le temps de réponse dépasse ce seuil. Dans la stratégie proposée dans (Tos et al., 2016), la réplication n'est considérée que si (i) le temps de réponse, estimé avant l'exécution d'une requête, est supérieur à un seuil de temps de réponse établi dans le SLA puis, (ii) la réplication doit être profitable si une réplication est envisagée. La stratégie proposée dans (Gill and Singh, 2016) vise à créer une réplique pour les données dont la popularité dépasse un certain seuil. Grâce au concept de 'sac à dos', les répliques sont transférées des CDs plus chers vers des CDs moins chers afin de réduire le coût de la réplication. Enfin, la stratégie proposée dans (Mansouri et al, 2017) ne réplique qu'une petite partie (20%) des données fréquemment demandées sur les meilleurs emplacements en fonction des requêtes des utilisateurs et de l'espace de stockage disponible.

En général, les stratégies statiques sont simples à mettre en œuvre et le choix d'une stratégie statique dépend notamment de trois facteurs importants : la stabilité du mode

d'accès des utilisateurs, la capacité de stockage des nœuds et la bande passante disponible. La création statique d'un nombre maximal de répliques peut garantir les performances requises au prix d'un coût de fonctionnement élevé (Lin et al., 2013). Néanmoins, les modes d'accès aux données varient énormément dans les systèmes Cloud. En plus, la charge de travail et la bande passante sont très hétérogènes. En conséquence, les stratégies dynamiques sont souhaitables en raison des aspects dynamiques des systèmes Cloud. Toutefois, elles présentent certains inconvénients, tels que la difficulté de collecter des informations d'exécution précises de tous les nœuds du système.

Stratégies centralisées vs décentralisées. En plus de la nature de la réplication, les stratégies de réplication ont aussi été classées suivant le mécanisme du contrôle de la réplication dans le système.

Une réplication centralisée est facile à mettre en œuvre vu qu'il n'y a qu'une seule entité qui prend la décision de réplication (Lei et al., 2008; Sun et al.,2012a; Huang et al., 2014; Zhang et al., 2018; Begum and Sirisha, 2019). La stratégie proposée dans (Zhang et al., 2018) s'appuie sur un nœud central qui maintient à jour une vue globale sur les données. Cela permet de réagir rapidement à l'évolution du réseau et à la variation de la charge de travail. Néanmoins, l'entité centrale doit avoir une connaissance globale de tous les paramètres liés à la réplication sur tous les nœuds du système. De plus, la présence d'une autorité centrale n'est pas idéale pour la fiabilité et la tolérance aux pannes.

Une réplication décentralisée remédie à ces problèmes. L'approche décentralisée (Ghemawat et al., 2003; Shvachko et al., 2010; Wei et al., 2010; Tos et al., 2018; Mansouri and Buyya, 2019) est un gage de fiabilité, car il n'existe pas de point de défaillance unique dans le système. Le système se comporte de manière prévisible, y compris si un certain nombre de nœuds sont déconnectés ou en panne. Cependant, l'absence de contrôle central ou la prise en compte d'informations incomplètes sur l'état du système peut conduire à des problèmes tels qu'une réplication excessive.

Dans ce qui suit, nous nous basons sur des critères, parfois spécifiques aux environnements Cloud, afin de proposer une nouvelle classification de ces stratégies. Nous classons ces stratégies suivant les critères suivants : (i) l'orientation du bénéfice (Tabet et al., 2015), (ii) la fonction objectif pour laquelle une stratégie a été conçue, (iii) le nombre d'objectifs qu'une stratégie de réplication vise à satisfaire, (iv) la nature de l'environnement, en termes de nombre de fournisseurs, pour lequel une stratégie a été conçue (Mokadem and Hameurlain, 2020), (v) la prise en compte des coûts économiques liés à la réplication de données (Tos et al., 2018; Mokadem et al., 2020). Bien entendu, cette classification peut engendrer un certain chevauchement. En d'autres termes, certaines stratégies peuvent d'être citées dans différentes classes.

3.4.1 Orientation du profit

L'objectif d'un fournisseur Cloud est de maximiser son bénéfice, tandis que l'objectif d'un locataire est de toujours obtenir un service maximal à un faible coût. On pourra alors classer les stratégies de réplication suivant le fait qu'elles visent : (i) à augmenter le profit du fournisseur ou (ii) à réduire les coûts monétaires payés par les locataires au fournisseur. On parle alors de stratégies de réplication orientées fournisseur vs. orientées locataires respectivement.

Dans la littérature, la plupart des stratégies de réplication proposées dans les systèmes Cloud sont axées sur la perspective de réduire la consommation de ressources pour les fournisseurs tout en satisfaisant les objectifs des locataires (Wei et al., 2010; Bonvin et al., 2011; Sousa and Machado, 2012; Zeng and Verravalli, 2014; Boru et al., 2015; Tos et al., 2016; Liu et al., 2018). On parle alors d'un profit orienté fournisseur du moment que celui-ci vise à réduire ses frais généraux. Ceci permet alors d'augmenter son profit économique, même si une grande partie de ces stratégies ne s'intéressent pas au profit monétaire du fournisseur.

La stratégie proposée dans (Sousa and Machado, 2012) vise à ajuster la consommation des ressources dans un environnement BD multi-locataires. En s'adaptant à la variation de la charge de travail dans les nœuds, les requêtes utilisent des répliques situées sur des nœuds moins chargés. Ceci permet de répondre aux objectifs des locataires tout en minimisant le coût monétaire des pénalités. La stratégie proposée dans (Tos et al., 2016) vise à réduire la consommation de la bande passante. De plus, elle vise à réduire les pénalités en créant une nouvelle réplique seulement si le SLA n'est pas satisfait. La stratégie proposée dans (Mansouri et al., 2017) vise également à augmenter le profit du fournisseur à travers une réduction significative de la consommation des ressources de stockage.

D'un autre coté, seulement quelques travaux de réplication de données s'intéressent à la réduction des coûts payés par les locataires au fournisseur (Sakr et al., 2011 ; Sharma et al., 2011; Sakr and Liu, 2012; Magalhaes and Silva, 2013; Zhao et al., 2015; Limam et al., 2018). On parle alors de stratégies dont le profit est orienté pour le locataire.

Le système de gestion de ressources Kingfisher (Sharma et al., 2011) permet à un locataire d'optimiser ses ressources en choisissant la configuration du serveur qui lui correspond le mieux du point de vue des performances et de coûts. Kingfisher réduit considérablement le temps de transition vers une nouvelle configuration dans le Cloud. Les auteurs dans (Sakr and Liu, 2012) proposent un outil d'approvisionnement en définissant de manière déclarative des règles spécifiques à l'application. Les ressources sont provisionnées de manière adaptative suivant les besoins du consommateur. La stratégie proposée dans (Magalhaes and Silva, 2013) collecte des données au niveau de l'application et détecte les anomalies de performances résultant de la variation de la charge de travail ou de la consommation de ressources. La stratégie proposée dans (Zhao et al., 2015) permet un approvisionnement dynamique et adaptatif des ressources, incluant des répliques de données, en se basant sur des règles définies par les applications. Elle permet de satisfaire les exigences de performances des locataires tout en en réduisant le coût monétaire des ressources allouées au locataire. Enfin, l'allocation des ressources aux locataires se fait suivant le budget initial de chaque locataire dans la stratégie proposée dans (Limam et al., 2018). La création d'une nouvelle réplique n'est envisagée qu'en cas de violation du SLA. De plus, le coût de la réplication ne doit, en aucun cas, dépasser un budget initial spécifique à la réplication.

3.4.2 Fonction objectif visée

Comme pour les stratégies de réplication proposées dans les grilles de données, une stratégie proposée dans les systèmes Cloud vise souvent à privilégier un comportement qui améliore les performances du système. Cela revient à satisfaire une fonction objectif. Nous pourrons alors classer ces stratégies suivant la fonction objectif visée. Nous distinguons :

(i) Les stratégies qui visent à améliorer la localité de données. Dans certaines stratégies (Abad et al., 2011; Lee et al., 2015), les données populaires sont répliquées au plus près des nœuds qui génèrent le plus de demandes, en supposant que ces données resteront probablement populaires à l'avenir (Jayalakshmi and Ranjana, 2015). On parle alors d'une localité temporelle et géographique. D'autres stratégies proposent des mécanismes pour prédire l'accès futur sur la base d'historiques d'accès afin de répliquer de manière préventive les données (He et al., 2018). Enfin, d'autres travaux exploitent une localité dite 'spatiale' en répliquant les données proches des données accédées récemment. Pour faire face à la vague de popularité rapide, la stratégie proposée dans (Ridhawi et al., 2015) prend en compte la variation de la popularité des données qui culmine pendant une courte période puis diminue progressivement.

(ii) Les stratégies qui visent à améliorer la disponibilité de la bande passante du réseau. Ces stratégies visent à réduire les transferts de données en répliquant sur des nœuds qui ont une meilleure bande passante avec le nœud qui demande les données (Ardekani and Terry, 2014; Liu et al., 2016; Tos et al., 2018; Tabet et al., 2018). Un placement au sein du centre de données local est privilégié. La stratégie proposée dans (Mokadem and Hameurlain, 2020) exploite la hiérarchie du point de vue de la bande passante du réseau en plaçant les nouvelles répliques dans la même région que les nœuds qui exécutent la requête du locataire. Ceci réduit les coûts liés au transfert de données.

(iii) Les stratégies de réplication qui visent à réduire le coût de la réplication en se basant sur un modèle de coûts mathématique (Bonvin et al., 2011; Xiong et al., 2011; Pu et al., 2015; Mansouri et al., 2017). Ce modèle prend en compte certains paramètres comme les statistiques d'accès aux données et la taille des répliques afin de réduire la consommation de ressources en termes de bande passante et de stockage respectivement. La stratégie proposée dans (Xiong et al., 2011) se base sur une technique d'apprentissage. Elle modélise le profit potentiel correspondant à l'exécution d'une requête sous différentes allocations de ressources afin de minimiser les pénalités payées par le fournisseur. Les auteurs dans (Calheiros et al., 2015) ont proposé un modèle de prédiction de la charge de travail dans le Cloud pour les fournisseurs de SaaS. Ce modèle peut répondre à la QdS avec une quantité rentable de ressources en estimant les besoins futurs. Dans la stratégie proposée dans (Tos et al., 2016), la décision de réplication repose sur un modèle de coûts qui estime le temps de réponse d'une requête avant son exécution. Cette estimation prend en compte certaines statistiques telles que la sélectivité. Les dépenses du fournisseur sont également estimées lorsqu'une réplication est envisagée.

(iv) Les stratégies de réplication qui visent à réduire le coût de la réplication en se basant sur des comportements économiques utilisés dans le commerce. Certaines de ces stratégies considèrent les données comme des biens, les requêtes comme des clients et les nœuds comme des entreprises. La stratégie proposée dans (Marcus et al., 2018) vise l'équilibrage de charge entre les nœuds d'un système. La valeur des biens est estimée en fonction de la fréquence d'accès à ces données et des frais monétaires que l'utilisateur paie pour l'exécution de ses requêtes. D'autres stratégies se basent sur le mécanisme de vente aux enchères (Belalem et al., 2011; Zhang et al., 2014; Fu et al., 2014; Shi et al., 2017). Dans (Zhang et al., 2014), une vente aux enchères est lancée pour déterminer l'emplacement d'une nouvelle réplique si le niveau de disponibilité souhaité ne peut pas

être maintenu. Le prix des enchères dépend de la probabilité de défaillance d'un nœud, de la bande passante du réseau et de l'espace de stockage disponible. Enfin, la stratégie proposée dans (Shi et al., 2017) présente des mécanismes d'enchères en ligne compétitifs pour l'approvisionnement à la demande et la tarification des MVs. A noter que seules certaines stratégies citées ci-dessus intègrent la dimension de rentabilité pour le fournisseur dans la fonction objectif. Ce critère est abordé dans la sous section 3.4.5.

3.4.3 Nombre d'objectifs visés

De nombreuses stratégies de réplication proposées dans la littérature visent à garantir un seul objectif pour les locataires, par exemple, l'équilibrage de charge (Wei et al., 2010), la fiabilité (Li et al., 2017), les performances en termes de latence (Ma et al., 2017) ou de temps de réponse (Tos et al., 2016), la sécurité (Ali et al., 2018) et la réduction de consommation énergétique (Boru et al., 2015). D'autres stratégies visent à prendre en compte le compromis entre différents objectifs.

3.4.3.1 Stratégies mono-objectif

Bien que l'amélioration des performances constitue l'objectif principal (coté locataires) de nos travaux de recherche, nous citons dans ce qui suit, d'autres objectifs visés par quelques stratégies. Nous nous concentrons uniquement sur les objectifs intégrés dans le SLA :

(i) ***Disponibilité.*** La disponibilité de données constitue l'objectif visé par la plupart des stratégies de réplication existantes dans la littérature (Myint and Naing, 2011; Sun et al., 2012a ; Gill and Singh, 2016; Mansouri, 2016; Liu et al., 2020). La disponibilité de données dépend souvent de la disponibilité des nœuds qui les hébergent. Un nœud surchargé ne peut tout simplement pas répondre à temps aux demandes des locataires. En conséquence, il faut souvent augmenter le nombre de répliques afin que les données demandées soient disponibles sur des nœuds moins chargée. Néanmoins, les auteurs dans (Wei et al., 2010) soulignent que le fait d'avoir trop de répliques n'augmente pas nécessairement la disponibilité, mais entraîne plutôt une baisse des performances. Alors, un nombre minimum de répliques est calculé et maintenu pour satisfaire un niveau de disponibilité minimum. L'étude de la relation entre le nombre de répliques et le niveau de disponibilité a également fait l'objet d'autres travaux (Xiong et al., 2011). La stratégie proposée dans (Qu and Xiong, 2012) vise à obtenir une haute disponibilité tout en maintenant un faible coût de réplication. Elle permet d'adapter le nombre de répliques en fonction du trafic réseau. Si une unité de données devient populaire, davantage de répliques sont alors créées ou déplacées.

(ii) ***Tolérance aux pannes/fautes.*** En cas de défaillance d'un nœud, les répliques de données qui s'y trouvent dans d'autres nœuds facilitent la récupération des données perdues. De nombreuses stratégies de réplication visent l'amélioration de la tolérance aux pannes (Chen et al., 2014; Vijayakumar et al., 2015; Selvi et al., 2018).

(iii) ***Fiabilité.*** La création de plusieurs répliques réduit la probabilité de perte de données. Dans la stratégie proposée dans (Li et al, 2011), une réplication incrémentielle est appliquée dans les CDs tout en respectant l'exigence de fiabilité des données. La stratégie proposée dans (Sun et al., 2012b) améliore la fiabilité en modélisant mathématiquement les taux de défaillance des nœuds du système.

(iv) ***Réduction de la consommation d'énergie.*** Certaines stratégies de réplication de données visent à réduire la consommation d'énergie (Long et al., 2014; Xu et al., 2015b; Boru et al., 2015; Zhang et al, 2015 ; Alghamdi et al., 2017; Edwin et al., 2019), tandis que d'autres stratégies tentent de réduire l'empreinte carbone en répliquant dans un CD plus écologique (Xu et al., 2015b).

(v) ***Objectif de performances.*** Peu de stratégies de réplication intègrent l'objectif de performances, notamment en termes de temps de réponse, dans le SLA (Bonvin et al., 2010a; Sousa and Machado, 2012; Bai et al., 2013; Kumar et al., 2014; Gill and Singh., 2016; Liu and Shen, 2017; Mansouri et al., 2017; Dabas and Aggarwal, 2019; Mansouri et al., 2019). Certaines stratégies tirent profit de la hiérarchie au niveau de la bande passante afin de réduire les coûts de communication (Tos et al., 2018). En conséquence, le temps de réponse des requêtes est réduit.

Il est utile de noter qu'une amélioration des performances, par exemple en termes de temps de réponse, peut être observée en conséquence d'autres objectifs. Les auteurs dans (Wei et al., 2010) affirment que les performances peuvent être améliorées en équilibrant la charge entre différent nœuds. Il en est de même pour la stratégie proposée dans (Lee et al., 2015). Favoriser la localité des données réduit également la consommation de la bande passante du réseau, ce qui améliore les performances du système (Mengxing et al., 2013; Xu et al., 2015a; Kloudas et al., 2015; Sharov et al., 2015; Vulimiri et al., 2015). Dans la stratégie proposée dans (Sharov et al., 2015), une réplique de chaque fragment de données est élue en tant que leader qui coordonne les tâches de réplication. Elle est responsable des opérations de mise à jour. Les auteurs affirment que leur stratégie améliore jusqu'à 50% la latence d'accès aux données.

Par ailleurs, certaines stratégies de réplication de données s'appuient sur d'autres techniques afin d'améliorer les performances. Des exemples de ces techniques sont : la mise en cache des données (Silvestre et al., 2012), la déduplication (Nicolae, 2015), la pré-lecture de données (Prefetching) (Mansouri and Javidi, 2018b), la migration de données (Mansouri and Buyya, 2019), l'accès en parallèle aux données (Mansouri et al., 2017), les techniques de 'Data Mining' (Slimani et al., 2020), l'apprentissage supervisé (Bui et al., 2016), la prédiction de surcharge des nœuds (Sun et al., 2018), le partitionnement (Kumar et al., 2014; Zhou and Fan, 2017), la fragmentation (Ali et al., 2018) et les codes correcteurs (Bui et al., 2016; Mao et al., 2016).

3.4.3.2 Stratégies multi-objectifs

Un certain nombre de stratégies visent la prise en compte du compromis entre plusieurs objectifs (Hassan et al., 2009; Long et al., 2014; Boru et al., 2015; Pu et al., 2015; Casas et al., 2017; Dai et al., 2017; Mansouri and Javidi, 2018a; Sun et al., 2018; Kaseb et al., 2019; Edwin et al., 2019; Mokadem and Hameurlain, 2020). La stratégie proposée dans (Hassan et al., 2009) vise de maintenir le stockage au dessous de certaines limites tout en essayant de trouver un compromis avec l'objectif de minimiser la latence et d'optimiser la fiabilité. La stratégie proposée dans (Long et al., 2014) se base sur des modèles mathématiques pour résoudre le compromis entre la disponibilité, le temps de réponse, la latence du réseau et l'équilibrage de charge. Une fonction à minimiser est proposée pour chaque objectif pondéré par le poids voulu par l'administrateur. Puis, une fonction globale regroupe l'ensemble de ces fonctions. Le placement des données peut être effectué selon une fonction d'optimisation prenant en compte le compromis entre ces objectifs.

Dans la stratégie proposée pour le système Iridium (Pu et al., 2015), la fréquence d'interrogation et les statistiques d'accès aux données sont utilisées lors du placement des répliques. La consommation de la bande passante entre les CDs est alors réduite, ce qui permet de minimiser la latence. La stratégie proposée dans (Boru et al., 2015) vise à réduire aussi bien la consommation énergétique que l'utilisation de la bande passante du réseau. La stratégie proposée dans (Li et al., 2017) réduit les coûts de stockage et de bande passante. La stratégie proposée dans (Dai et al., 2017) permet d'assurer un équilibrage de charge tout en réduisant le temps de réponse. La stratégie EIMORM (Edwin et al., 2019) se base sur le concept de 'sac à dos'. Elle vise à équilibrer entre plusieurs objectifs tels que la disponibilité de données, l'équilibrage de la charge et le coût de la réplication. Enfin, la stratégie proposée dans (Mokadem and Hameurlain, 2020) vise à réduire le temps de réponse et à assurer une disponibilité minimum de données.

3.4.4 Nature de l'environnement Cloud (nombre de fournisseurs)

Bien que la plupart des stratégies de réplication citées ci-dessus ont été proposées dans un environnement de Cloud mono-fournisseur, quelques stratégies ont été déployées dans un environnement Cloud multi-fournisseurs (Abu-Libdeh et al., 2010; Bessani et al., 2011; Bessani et al., 2013; Chen et al., 2014; Wu et al., 2013; Abouzamazem and Ezhilchelvan, 2013; Grozev and Buyya, 2015; Liu and Shen, 2017; Mansouri and Buyya, 2019). Nous utilisons alors le nombre de fournisseurs comme un critère de classification de ces stratégies.

L'une des premières stratégies de réplication déployée entre différent fournisseurs de Cloud a été proposée dans le système DepSky (Bessani et al., 2011). Cependant, elle ne porte que sur les aspects de sécurité sans prendre en compte les aspects économiques du Cloud. La stratégie de réplication de données intégrée au système SpanStore (Wu et al., 2013) couvre plusieurs fournisseurs de Cloud. L'écart de prix entre ces fournisseurs est exploité afin de minimiser les coûts lors de la prise en compte des exigences de tolérance aux pannes et de latence. Dans la stratégie proposée dans (Abouzamazem and Ezhilchelvan, 2013), les locataires louent les services de plusieurs fournisseurs suivant une politique tarifaire prenant en compte les prix de ressources fournis par chaque fournisseur de Cloud. Les auteurs dans (Mansouri and Buyya, 2019) se sont concentrés sur les coûts de l'opération Get/Put lors d'une réplication. Comme dans (Miglierina et al., 2013), l'hétérogénéité des ressources et la différence de prix de ces ressources entre différents fournisseurs ont été exploitées pour minimiser le coût monétaire de la réplication.

La stratégie proposée dans (Liu and Shen, 2017) exploite des techniques de programmation dynamique et non linéaire pour maximiser la disponibilité des données en cas de panne d'un nœud et minimiser le coût de la réplication. Elle permet également de remédier au problème de verrouillage du fournisseur (un locataire n'est souvent pas libre de passer d'un fournisseur à un autre). Enfin, la stratégie de réplication des données proposée dans le système RACS (Abu Libdeh et al., 2010) récupère les données d'un Cloud qui est sur le point de tomber en panne et les déplace vers un nouveau Cloud.

3.4.5 Prise en compte des coûts économiques

Dans cette section, nous considérons la prise en compte des coûts économiques liés à la réplication de données comme un critère de classification des stratégies de réplication de données dans les systèmes Cloud. Nous nous intéressons particulièrement à la prise en

compte du profit économique, et notamment monétaire, du fournisseur ainsi qu'à la prise en compte de la consommation énergétique par le fournisseur lors de la réplication. Dans ce qui suit, nous nous focalisons uniquement sur les stratégies visant à satisfaire l'objectif des performances.

3.4.5.1 Stratégies ne prenant pas en compte le coût économique de la réplication

La plupart des stratégies proposées dans la littérature ne s'intéressent qu'à la réduction des ressources utilisées (par exemple en termes de ressources de stockage et/ou de transfert de données entre CDs) lors de la réplication de données sans pour autant prendre en compte les coûts économiques de cette réplication (Wei et al., 2010; Moon et al. 2013; Kumar et al., 2014; Long et al., 2014; Pu et al., 2015; Tan and Babu 2016; Mansouri et al., 2017; Mansouri and Javidi, 2018b; Sun et al., 2018; Dabas and Aggarwal, 2019).

La réplication adoptée dans (Elmore et al., 2009) vise à réduire les ressources en termes de bande passante lors de la migration des données pour le placement de répliques dans un SGBD à locataires multiples. Dans la stratégie proposée dans (Moon et al. 2013), les performances sont améliorées via l'équilibrage de charge. Cependant, cette stratégie se concentre sur la consolidation du plus grand nombre possible de locataires dans un même matériel sans prendre en compte les violations du SLA. La stratégie proposée dans (Kumar et al., 2014) minimise le nombre de nœuds impliqués lors de l'exécution d'une requête. Elle permet de réduire la consommation de ressources sans pour autant chiffrer cette consommation du point de vue économique. Les auteurs dans (Tan and Babu 2016) spécifient de manière déclarative les objectifs de performances dans le contexte de BD parallèles multi-locataires dans des systèmes tels que Hadoop ou Spark, sans pour autant prendre en compte les coûts économiques. Enfin, les auteurs dans (Dabas and Aggarwal, 2019) affirment que le temps d'accès aux données est réduit à l'aide d'un seuil dynamique de temps de réponse. La réplication est faite de manière différée sans prise en compte des coûts économiques.

3.4.5.2 Stratégies prenant en compte le coût économique de la réplication

Certaines stratégies sont mentionnées comme prenant en compte le coût économique de la réplication de données. Néanmoins, le modèle de coûts proposé n'intègre pas nécessairement un coût monétaire (Ananthanarayanan et al., 2011; Janpet and Wen, 2013; Lin et al., 2013; Calheiros et al., 2015; Gill and Singh, 2016, Marcus et al., 2018; Edwin et al., 2019).

Skute (Bonvin et al., 2010b) est une stratégie basée sur une économie virtuelle. Les MVs annoncent périodiquement leurs locations aux autres MVs. Elles accumulent de la richesse en répondant aux requêtes. Puis, elles dépensent cette richesse en stockant des répliques sur des ressources spécifiques à d'autres MVs en fonction de leur loyer. Skute vise à minimiser les coûts de communication tout en maximisant le profit économique virtuel. Les auteurs dans (Xiong et al., 2011) utilisent un modèle prédictif pour l'allocation de CPU et de mémoire. Ils utilisent des techniques d'apprentissage automatique pour générer un montant optimal de profit virtuel dans un environnement de BD multi-locataires. La stratégie utilisée dans Scarlett (Ananthanarayanan et al., 2011) se base sur un historique des réplications afin de limiter la surcharge des nœuds. Elle calcule un facteur de réplication pour chaque fichier et crée des répliques à budget virtuel. Dans la stratégie proposée dans (Lin et al., 2013), le coût associé à la réplication est directement proportionnel au nombre de répliques.

Néanmoins, ce coût est modélisé en termes de temps. Il en est de même dans la stratégie DCR2S proposée dans (Gill and Singh, 2016). Une réplication n'est possible que si le coût de celle-ci ne dépasse pas une valeur budgétaire assignée initialement au CD. Enfin, le coût de réplication considéré dans (Edwin et al., 2019) n'est pas un coût monétaire réel. Il est considéré comme une valeur attribuée aux CDs.

Seules quelques stratégies prennent en compte les aspects économiques en modélisant des coûts monétaires dans leur modèle de coûts. Dans ce qui suit, nous nous focalisons sur : (i) les stratégies qui modélisent les coûts liés à la réplication dans les dépenses monétaires du fournisseur. Certaines de ces stratégies prennent également en compte le profit monétaire du fournisseur, et (ii) les stratégies de réplications qui visent à réduire les dépenses du fournisseur du point de vue de la consommation énergétique.

(i) Peu de stratégies modélisent le coût de la réplication dans les dépenses monétaires du fournisseur tout en satisfaisant l'objectif de performances pour les locataires (Wu et al., 2013; Zeng and Veeravalli, 2014; Zeng et al., 2016; Casas et al., 2017; Tos et al., 2018; Liu and Shen, 2017; Mansouri and Buyya, 2019; Mokadem and Hameurlain, 2020). Parmi ces stratégies, très peu s'intéressent au profit monétaire du fournisseur.

La stratégie proposée dans le système SpanStore (Wu et al., 2013) prend en compte les écarts de prix monétaires des ressources entre différents fournisseurs, permettant de minimiser le coût monétaire de la réplication. La stratégie proposée dans (Mansouri and Buyya, 2019) tire également profit de la tarification hétérogène entre CDs, en termes de stockage et de bande passante, en répliquant suivant la variation future de la charge de travail. La stratégie proposée dans (Zeng et al., 2016) prend en compte le compromis entre le temps d'exécution et le coût monétaire de la réplication des métadonnées dans chaque nœud, tout en maximisant l'utilisation des ressources de stockage. Le nombre de répliques et leur placement dépendent également de ce compromis.

Dans la stratégie proposée dans (Tos et al., 2018), les dépenses et les revenus monétaires du fournisseur sont estimées avant l'exécution d'une requête si une réplication est envisagée. Une réplique est créée seulement si cette réplication est profitable pour le fournisseur. De plus, l'emplacement des répliques est basé sur la sélection du CD le moins cher qui satisfait l'objectif de temps de réponse. Dans la stratégie proposée dans de domaine des flux de travaux (Workflows) (Casas et al., 2017), une réplication est envisagée seulement si le coût monétaire d'une application ne dépasse pas une limite monétaire fixée à l'avance dans le SLA. Enfin, une réplication doit être profitable pour le fournisseur dans la stratégie proposée dans (Mokadem and Hameurlain, 2020). Les coûts engendrés par la réplication et les pénalités éventuelles sont incluses dans les dépenses monétaires du fournisseur.

(ii) La plupart des stratégies de réplication de données citées ci-dessus négligent la puissance et l'énergie nécessaires à la réplication de données. Or, les CDs consomment une part importante de la consommation mondiale d'électricité et les factures d'énergie sont devenues le deuxième plus gros coût dans les budgets des fournisseurs de Cloud (Zakarya et Gillam, 2017). Cela est dû au développement d'Internet et à l'augmentation des besoins de stockage dans les entreprises (Mastelic et al., 2014). Certaines études (Shehabi et al., 2016) prévoient que la consommation énergétique mondiale augmentera de 13% en 2030 à cause de la consommation d'énergie dans les CDs.

	Classification proposée								
	Statique vs. Dynamique	Centralisée vs. Décentralisée	orientée fournisseur vs. locataire	Fonction objectif considérée	Mono vs. Multi Objectifs	Mono vs. Multi fournisseurs	considération aspects économiques	coûts monétaires	Consom. énergétique
Ghemawat et al., 2003	X	X	X	Localité données	Mono	Mono	-	-	-
Hassan et al., 2009	X	X	X	Localité données	Multi	Mono	-	-	-
Abu-Libdeh et al., 2010	X	X	X	Localité données	Multi	Multi	-	-	-
Wei et al., 2010	X	X	X	Modèle de coûts	Multi	Mono	-	-	-
Bonvin et al., 2010b	X	X	X	Localité données.	Mono	Mono	X	-	-
Xiong et al., 2011	X	X	X	Modèle de coûts	Mono	Mono	X	X	-
Sun et al., 2012a	X	X	X	Modèle de coûts	Mono	Mono	-	-	-
Sakr and Liu, 2012	X	X	X	Modèle de coûts	Mono	Mono	-	-	-
Bai et al., 2013	X	X	X	Localité données	Multi	Mono	-	-	-
Lin et al., 2013	X	X	X	Modèle de coûts	Multi	Mono	-	-	-
Wu et al., 2013	X	X	X	Localité réseau	Multi	Multi	X	X	-
long et al., 2014	X	X	X	Localité données	Multi	Mono	-	-	X
Zeng and Veeravalli, 2014	X	X	X	Modèle de coûts	Multi	Mono	X	-	-
Kumar et al., 2014	X	X	X	Localité données	Multi	Mono	-	-	X
Boru et al., 2015	X	X	X	Localité données	Multi	Mono	-	-	X
Zhao et al., 2015	X	X	X	-	Multi	Mono	X	-	-
Gill and Singh, 2016	X	X	X	Localité données	Multi	Mono	X	-	-
Zeng et al., 2016	X	X	X	Modèle de coûts	Mono	Mono	X	X	-
Mansouri et al., 2017	X	X	X	Localité données	Multi	Mono	-	-	
Tos et al., 2018	X	X	X	Localité réseau	Multi	Mono	X	X	-
Shi et al., 2017	X	X	X	Comp. Economiq.	Multi	Multi	X	X	-
Liu and Shen, 2017	X	X	X	Modèle de coûts	Multi	Multi	X	X	-
Alghamdi et al., 2017	X	X	X	Modèle de coûts	Multi	Mono	X	X	X
Mansouri and Javidi, 2018b	X	X	X	Localité données	Multi	Mono	-	-	-
Sun et al., 2018	X	X	X	Modèle de coûts	Multi	Mono	-	-	-
Liu et al., 2018	X	X	X	-	Multi	Mono	X	-	-
Mansouri and Buyya, 2019	X	X	X	Modèle de coûts	Multi	Multi	X	X	-
Limam et al., 2019	X	X	X	Localité données	Multi	Mono	X	X	-
Edwin et al., 2019	X	X	X	Modèle de coûts	Multi	Mono	X	-	X
Mokadem and Hameurlain, 2020	X	X	X	Localité réseau	Multi	Mono	X	X	-

TABLE 3.1 - Classification de certaines stratégies de réplication dans les systèmes Cloud.

Quelques stratégies de réplication de données se sont intéressées à la consommation énergétique (Xu et al., 2015). Dans ce qui suit, nous nous intéressons aux stratégies de

réplication de données qui visent à réduire la consommation énergétique tout en prenant en compte le profit du fournisseur.

La stratégie proposée dans (Zhang et al., 2015) vise à regrouper la charge de travail en plaçant des répliques seulement sur quelques nœuds afin de pouvoir mettre en veille ou éteindre les nœuds inactifs. La stratégie proposée dans (Boru et al., 2015) modélise la consommation énergétique et l'utilisation de la bande passante. Si le nombre d'accès aux données dépasse un certain seuil, une réplication est faite sur le CD qui consomme le moins d'énergie et de bande passante. Au lieu de répliquer les données sur la BD centrale, elles sont répliquées sur des nœuds proches des serveurs de calculs. La stratégie proposée dans (Alghamdi et al., 2017) prend en compte la consommation d'énergie afin de réduire le coût monétaire d'une requête utilisateur. Les auteurs ont affirmé que l'algorithme proposé réduit la consommation d'énergie d'au moins la moitié par rapport à une solution de réplication optimale.

Dans (Séguéla et al., 2019a, Séguéla et al., 2019b), nous avons comparé différentes stratégies de réplication du point de vue de la consommation énergétique. Les stratégies comparées sont : la stratégie proposée dans (Boru et al., 2015), la stratégie MORM (long et al., 2014) visant à réduire la consommation énergétique et la stratégie PEPR proposée dans (Tos et al., 2016), conçue initialement pour la prise en compte du profit du fournisseur et que nous avons étendu afin de modéliser la consommation énergétique au sein des dépenses du fournisseur. L'évaluation des performances a montré que la stratégie proposée dans (Boru et al., 2015) réduit le coût du fournisseur au prix d'une consommation d'énergie plus élevée tandis que PEPR permet de réduire les coûts sans aucun effet sur la consommation énergétique. Néanmoins, un nombre de violations important peut être observé dans le cas où la réplication n'est pas profitable. À l'opposé, MORM consomme moins d'énergie au prix d'importantes dépenses dues à un placement initial d'un nombre élevé de répliques.

3.5 ANALYSE

Les fournisseurs de Cloud s'attendent à rentabiliser leurs relations commerciales avec les locataires tout en satisfaisant une certaine QdS pour ces derniers. En conséquence, une stratégie de réplication de données dans les systèmes Cloud doit non seulement répondre aux problématiques classiques auxquelles doit répondre toute stratégie de réplication de données mais, elle doit aussi prendre en compte les coûts économiques liés à la réplication.

Dans la littérature, de nombreuses stratégies de réplication ont été proposées dans les systèmes Cloud. Cependant, peu de stratégies visent à satisfaire les objectifs de performances pour les locataires en prenant en compte les aspects économiques tels que le profit monétaire du fournisseur. Néanmoins, une stratégie de réplication de données efficace dans les systèmes Cloud doit prendre en compte les compromis : (i) entre les différents objectifs conflictuels du locataire, par exemple les performances et la disponibilité, et (ii) entre la satisfaction de la QdS pour les locataires et le profit économique pour le fournisseur.

Pour une meilleure comparaison entre ces stratégies, nous avons identifié dans la Table 3.1, une liste non exhaustive de stratégies de réplication de données en se positionnant par rapport aux critères sur lesquels se basent les classifications existantes ainsi que la classification proposée dans les systèmes Cloud.

3.6 CONCLUSION

Les solutions apportées aux problèmes de performances dans les systèmes de grille de données, ne s'adaptent pas aux systèmes Cloud. Les stratégies de réplication de données n'échappent pas à cette règle. Les stratégies de réplication de données proposées dans les systèmes de grille de données visent à obtenir les meilleures performances possibles pour les utilisateurs. Cela passe par une utilisation maximale des ressources en créant autant de répliques que possible. Cela peut engendrer des coûts d'exploitation plus élevés que prévu et représente un gaspillage de ressources pour le fournisseur du Cloud. Une stratégie proposée dans les systèmes Cloud doit alors viser la satisfaction d'une QdS en termes d'objectifs pour les locataires sans chercher à aller au delà des attentes des locataires tout en prenant en compte le profit du fournisseur.

L'étude de l'état de l'art nous a permis de constater que la plupart des études dans la littérature ont classé ces stratégies comme étant : (i) statiques vs. dynamiques, et/ou (ii) centralisées vs. décentralisées. En prenant en compte des caractéristiques et critères, parfois spécifiques aux environnements Cloud, nous avons proposé de classer ces stratégies en se basant sur : (i) l'orientation du profit, (ii) la fonction objectif visée, (iii) le nombre d'objectifs SLO inclus dans le SLA avec un focus sur l'objectif des performances d'autant plus que la plupart des fournisseurs commerciaux n'incluent pas l'objectif de temps de réponse dans le SLA, (vi) le nombre de fournisseurs du Cloud, et enfin, (v) la prise en compte des coûts économiques liés à la réplication.

Aussi surprenant, nous avons constaté que la plupart des stratégies de réplication proposées dans la littérature ne s'intéressent qu'à la réduction de la consommation des ressources lors de la réplication sans pour autant se focaliser sur les coûts économiques de cette réplication. Pourtant, les systèmes Cloud tels que décrits par (Foster et al., 2008) sont basés sur l'économie. De plus, très peu de stratégies intègrent les coûts engendrés par la réplication, les pénalités payées aux locataires et les coûts de la consommation énergétique dans les dépenses monétaires du fournisseur. Enfin, une infime partie de ces stratégies intègrent l'objectif de temps de réponse dans le SLA tout en prenant en compte le profit économique (monétaire) du fournisseur. Ceci constitue une motivation pour nos travaux de recherche.

Dans le chapitre suivant, nous présentons nos propositions pour la gestion de la réplication de données en prenant en compte le compromis entre la satisfaction des performances pour les locataires, en termes de temps de réponse, et le profit économique (monétaire) pour le fournisseur.

CHAPITRE 4

Gestion de la réplication de données dans les systèmes Cloud

Résumé

Dans ce chapitre, nous présentons nos propositions permettant de répondre aux problèmes liés à la réplication de données dans les systèmes Cloud. D'abord, nous présentons l'architecture considérée qui s'articule autours de centres de données distribués à grande échelle. Puis, nous proposons un modèle de coûts sur lequel se base la prise de décision de réplication. Avant l'exécution de chaque requête d'un locataire, nous estimons le temps de réponse du plan d'exécution fourni par l'optimiseur du SGBD. La création d'une réplique est envisagée, le plus souvent par groupe de requêtes, uniquement si l'objectif de temps de réponse n'est pas satisfait. Nous proposons une heuristique permettant de trouver un placement acceptable d'une réplique d'une manière rentable. Pour cela, le modèle de coûts initial est étendu afin de prendre en compte les coûts économiques liés à l'exécution de la requête. Un modèle de coûts économique permet l'estimation des revenus et des dépenses (incluant les coûts de la réplication) du fournisseur lorsqu'une réplication est envisagée. Afin de permettre une gestion élastique des ressources, nous proposons également un algorithme d'ajustement dynamique du nombre de répliques. Enfin, nous présentons un algorithme qui évite le payement répétitif de pénalités par le fournisseur à ses locataires.

Sommaire

4.1 INTRODUCTION

Les locataires louent souvent les services d'un fournisseur de Cloud afin d'héberger des données de plus en plus volumineuses. Ils peuvent également payer aux fournisseurs

d'autres services tels que la possibilité de soumettre des requêtes permettant d'interroger efficacement ces données. Pour exécuter ces requêtes, un fournisseur alloue des ressources (de calcul, de réseau et de stockage) à chaque locataire. Pour cela, un certain nombre de nœuds, hébergés dans des Centres de Données (CD) souvent répartis à travers le monde, sont parfois nécessaires pour l'exécution de ces requêtes, d'autant plus qu'une certaine QdS est attendue par ces locataires. Tout au long de ce document, un nœud réfère à une Machine Virtuelle (MV) et un CD est composé de milliers de nœuds qui résident sur des hôtes physiques (Figure 4.1). Par ailleurs, un volume de stockage, des ressources propres de calcul (dans notre cas, une CPU) et une mémoire (spécifique à cette CPU) sont allouées à chaque MV, en plus de capacités réseau en termes de bande passante. Rappelons aussi que nous considérons un fournisseur de Cloud de type PaaS public.

Répondre aux attentes de performances des locataires sans sacrifier le profit économique du fournisseur est un défi difficile à relever pour les fournisseurs de Cloud. C'est pour cela que les propositions que nous détaillons dans ce chapitre doivent répondre aux problèmes classiques de la réplication de données tout en prenant en compte les coûts économiques liés à la réplication. Lorsqu'un locataire soumet une requête Q à un fournisseur, celle-ci est redirigée vers ses CDs afin qu'elle soit exécutée. Les problématiques liées à la réplication sont traitées de la manière suivante :

(i) Quand est ce que la réplication est envisagée ? Cela dépend de la satisfaction du SLA. Une création de réplique est envisagée uniquement si un objectif d'un locataire n'est pas satisfait.

(ii) Quelles données sont concernées par la réplication ? Une fois que la réplication est envisagée, il faudrait identifier la raison qui a causé la violation du SLA, ce qui permet d'identifier les données à répliquer**.**

(iii) Quel placement pour la nouvelle réplique ? Ce placement doit répondre aux objectifs du locataire de manière rentable pour le fournisseur. En d'autres termes, les coûts liés au placement d'une nouvelle réplique doivent être réduits autant que possible tout en satisfaisant les objectifs du locataire.

(iv) Combien de répliques sont créées ? Trop peu de répliques risquent de ne pas satisfaire l'objectif du locataire alors qu'un trop grand nombre de répliques augmente les coûts pour le fournisseur. Il faudrait alors ajuster le nombre de répliques de telle sorte que les objectifs des locataires soient satisfaits tout en étant profitables pour le fournisseur.

Nos propositions tiennent compte de l'hétérogénéité des ressources dans les systèmes Cloud. Cette hétérogénéité concerne les capacités matérielles, par exemple en termes de bande passante de réseau, de capacités de calcul et de stockage, ainsi que la tarification de ces ressources d'un CD à l'autre. Nous supposons que la tarification des ressources est négociée au préalable entre le fournisseur et ses locataires. Par ailleurs, nous considérons un Cloud qui exploite des CDs répartis géographiquement à grande échelle. Rappelons également que nous nous intéressons à la conception de stratégies dynamiques de réplication de données, visant l'amélioration des performances dans les systèmes de gestion de bases de données à grande échelle pour des applications décisionnelles, c'est-à-dire que les données sont en lecture seule. Dans ce qui suit, nous mettons en évidence l'architecture considérée du système Cloud. Puis, nous présentons nos propositions qui répondent aux problématiques citées ci-dessus.

4.2 ARCHITECTURE CONSIDEREE

Dans le chapitre précédent, nous avons vu qu'une stratégie de réplication est conçue en fonction de la topologie du système pour laquelle elle a été proposée, d'où l'impact de l'architecture sur les performances d'une telle stratégie.

Dans les systèmes parallèles tels que les systèmes Cloud, la répartition des relations sur plusieurs nœuds permet : (i) d'augmenter la bande passante en Entrées/Sorties (E/S) en exploitant au maximum la parallélisation des opérations de lecture/écriture d'une ou plusieurs relations, (ii) d'effectuer des traitements parallèles le plus proche possible des données et (iii) de favoriser l'équilibrage de charge pour améliorer les performances (Hameurlain et al., 1996).

La répartition de gros volume de données sur un seul CD/cluster lors de l'exécution d'une requête peut générer un transfert important de données. Les fournisseurs de Cloud doivent alors déployer ces données sur plusieurs CDs couvrant de vastes zones géographiques. Par exemple, les fournisseurs commerciaux de Cloud tels que Microsoft et Google, établissent souvent plusieurs installations dans des régions géographiques distinctes[18] [19]. Cela permet la fourniture de services en ligne afin de satisfaire les demandes de milliers, voir des millions, de locataires présents dans différents emplacements à travers le monde.

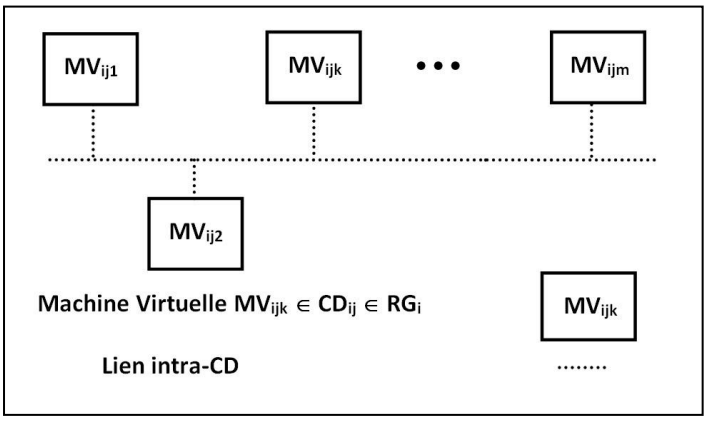

FIGURE 4.1 - Architecture d'un Centre de Données (CD).

Certaines solutions (Cooper et al., 2008; Ardekani and Terry, 2014; Tos et al., 2016) modélisent une hiérarchie de Cloud à deux niveaux. Dans ce cas, le système est vu comme des régions interconnectées. Chaque région est constituée d'un seul CD. Les stratégies de réplication de données visent alors à réduire la consommation de bande passante entre CDs (Kloudas et al., 2015; Liu et al., 2016). Cependant, les ressources de réseau entre les CDs sont hétérogènes. Dans la réalité, les régions, les CDs et les nœuds sont souvent interconnectés de manière hiérarchique. Cette hiérarchie concerne notamment la bande passante du réseau (Park et al., 2004). De plus, la présence d'installations dans plusieurs régions peut également faire varier les coûts de fonctionnement du fournisseur (par exemple, les coûts énergétiques et de réseau) d'une région à l'autre. Ainsi, la bande passante est relativement moins chère et beaucoup plus abondante à l'intérieur d'un CD, alors qu'elle est plus chère et moins

[18] Microsoft DCs. http://www.microsoft.com/en-us/server-cloud/cloud-os/global-datacenters.aspx

[19] Google DC Locations. http://www.google.com/about/datacenters/inside/locations/

abondante entre les régions où l'infrastructure Internet est habituellement utilisée. Par conséquent, nous considérons dans chaque région, plusieurs CDs communicants au travers d'une bande passante intermédiaire. Cela nous conduit à considérer une topologie à trois niveaux : régions, centre de données et enfin les nœuds qui hébergent les données (Figure 4.2).

Dans nos travaux, les relations d'une BD peuvent être fragmentées et distribuées sur de nombreux nœuds répartis dans des CDs, eux même situés dans différentes régions géographiquement distribuées. Les nœuds, CDs et régions sont interconnectés par des connexions réseau dont la capacité en termes de bande passante et de coût varient.

Soit $RG = \{RG_1, ..., RG_i, ..., RG_q\}$ avec $(1 \leq i \leq q)$ un ensemble de q régions géographiques connectées via Internet sans lien direct entre elles (Figure 4.2). La capacité de la bande passante disponible entre ces régions n'est pas abondante avec un prix plus ou moins élevé.

Nous utilisons $CD = \{CD_{i1}, ..., CD_{ij}, \ldots, CD_{in}\}$ avec $(1 \leq j \leq n)$ pour désigner un ensemble de n CDs hétérogènes dans une région RG_i. La bande passante entre ces CDs est plus abondante et moins chère que le premier niveau.

Enfin, nous utilisons $N = \{N_{ij1}, \ldots, N_{ijk}, ..., N_{ijm}\}$ avec $(1 \leq k \leq m)$ pour désigner un ensemble de m nœuds (MVs) dans chaque CD_{ij} comme le montre la Figure 4.1. Les nœuds sont interconnectés à travers une bande passante élevée dont le prix est encore moins cher que le prix de la bande passante entre CDs d'une même région.

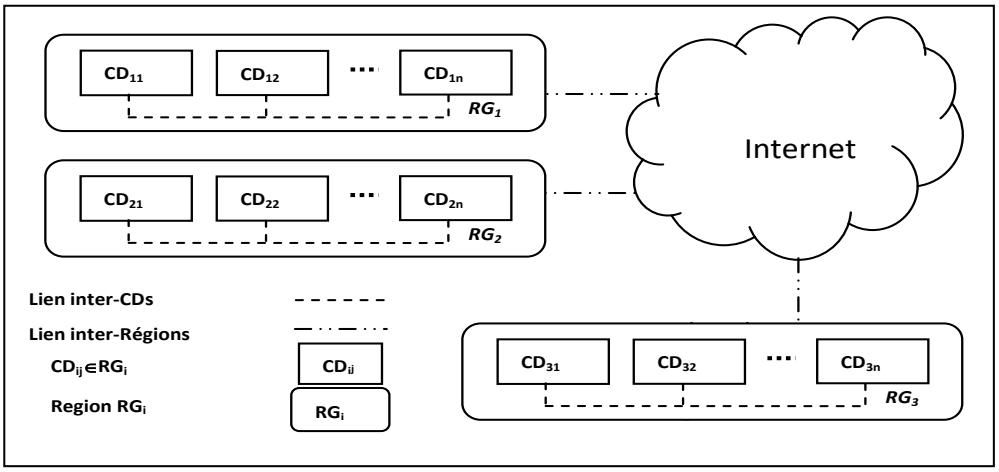

FIGURE 4.2 - Un exemple de centres de données distribués à grande échelle.

Comme exemple, nous considérons un Cloud composé de CDs distribués sur trois continents (RG_1, RG_2 *et* RG_3), comme illustré dans la Figure 4.2. Les CDs au sein d'une même région RG_i sont situés dans différentes villes. Ensuite, chaque CD_{ij} contient un certain nombre de MVs avec un espace de stockage et des ressources de calcul alloués pour chaque MV_{ijk} (N_{ijk}). Cela correspond à une architecture parallèle à mémoire distribuée (Shared-Nothing) comme illustré dans la Figure 4.3 (Ozsu and Valduriez, 2020). Les avantages d'une telle architecture par rapport à d'autres architectures telles que les architectures à mémoire partagée (Ranganathan et al., 1998) ou à disque partagé (Rahm and Stohr, 1995), sont la réduction des interférences résultant du partage des ressources et l'exploitation de la mémoire, indépendamment du réseau d'interconnexion. De plus, les architectures à mémoire

distribuée sont plus facilement extensibles et plus fiables par rapport aux autres architectures.

Du point de vue opérationnel, les locataires utilisent les services qu'ils louent au fournisseur. Leurs requêtes sont placées dans un environnement d'exécution parallèle. Nous supposons qu'une requête Q d'un locataire peut nécessiter des relations réparties dans différents CDs n'appartenant qu'à une seule région RG_i. Le gestionnaire d'allocation de ressources dans RG_i reçoit le plan d'exécution parallèle de la requête Q (fourni par l'optimiseur de requêtes du SGBD), puis effectue l'allocation de ressources. Il coordonne l'exécution de Q sur les nœuds de RG_i en affectant une tâche élémentaire, par exemple un opérateur relationnel, sur un nœud pourvu de ressources (CPU, mémoire et stockage). Il assure également le suivi des informations importantes telles que le nombre de répliques et leurs emplacements. Un gestionnaire de répliques au sein de chaque CD est responsable de la création/suppression des répliques. Enfin, le dictionnaire de données du SGBD (la métabase) est mis à jour à chaque fois qu'une réplique est créée/supprimée.

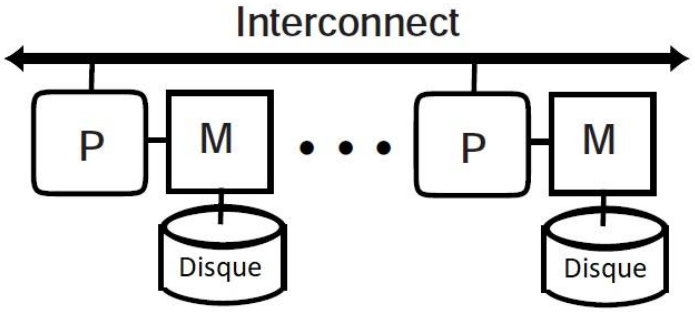

FIGURE 4.3 - Architecture parallèle à mémoire distribuée.

4.3 STRATEGIE DE REPLICATION DE DONNEES

Supposons qu'un locataire soumet une requête Q au fournisseur de Cloud. Celle-ci peut être composée de plusieurs opérateurs exécutés sur différents nœuds appartenant à RG_i. Lors de l'exécution de Q, les ressources allouées par l'optimiseur peuvent s'avérer insuffisantes pour répondre aux objectifs SLO du locataire, par exemple en termes de temps de réponse. Cela peut engendrer une violation du SLA. Différentes raisons peuvent être à l'origine de cette violation :

(i) Un transfert de relations réparties sur différents CDs, à travers une faible bande passante du réseau, peut entrainer une dégradation de la QdS, en termes de temps de réponse.

(ii) Un partage de ressources entre plusieurs locataires ou une requête complexe contenant de nombreuses dépendances entre ses différents opérateurs, peuvent également engendrer un manque de ressources de calcul à la suite d'une surcharge d'un nœud.

Afin de résoudre ces problèmes, la réplication de données peut être utile en vue de la satisfaction du contrat SLA, ce qui évite au fournisseur le payement de pénalités au locataire concerné. Les propositions que nous décrivons dans ce qui suit sont essentiellement issues des stratégies de réplication de données suivantes : (i) RSPC (*data Replication Strategy*

satisfying Performance tenant objective and provider profit in Cloud data centers) (Mokadem and Hameurlain, 2020) et (ii) PEPR (*PErformance and Profit oriented data Replication strategy*) (Tos et al., 2018). Ces propositions répondent aux problématiques liées à la réplication de la manière suivante :

(i) L'identification du moment de la réplication de données s'appuie sur des seuils de temps de réponse. Nous avons proposé un modèle de coûts qui permet de prédire si l'objectif de temps de réponse est satisfait avant l'exécution de *Q*. Cela passe par l'estimation du temps de réponse du plan d'exécution de *Q* fourni par le SGBD. Une création d'une réplique n'est envisagée que si l'objectif de temps de réponse n'est pas satisfait. Pour cela, nous avons proposé un algorithme qui décide de la création d'une nouvelle réplique. On verra plus loin qu'une réplique peut aussi être créée lors de la satisfaction de l'objectif de disponibilité minimum.

(ii) Les relations à répliquer doivent être correctement identifiées. Etant donné qu'une requête peut nécessiter plusieurs relations distribuées sur des nœuds appartenant à des CDs répartis dans une région donnée RG_i, il faudrait alors identifier les relations qui sont à l'origine de la violation du SLA.

(iii) Trouver un placement optimal pour une nouvelle réplique ne constitue pas une bonne solution à cause des coûts élevés lorsque l'ensemble des nœuds du système sont visités. La réduction de l'espace de recherche permet alors de répondre à cette problématique. Nous proposons une heuristique afin de trouver un placement acceptable qui satisfait les exigences du locataire tout en étant profitable du point de vue économique pour le fournisseur. Pour cela, le modèle de coûts initial est étendu. Le modèle de coûts obtenu, dit 'modèle de coûts économique', permet l'estimation des revenus et des dépenses du fournisseur pour l'exécution de *Q* lorsqu'une réplication est envisagée.

(iv) Le nombre de répliques créées pour chaque relation, doit être ajusté de manière dynamique en fonction de l'évolution de la demande des locataires. Pour cela, un autre algorithme permet de supprimer une réplique quand celle-ci n'est plus utile.

4.3.1 Conditions pour la réplication

Une stratégie doit être capable de décider s'il faut ou non créer une nouvelle réplique avant l'exécution d'une requête. Nous nous basons sur la satisfaction du SLA. Une nouvelle réplique est créée seulement dans les deux cas suivants :

(i) Initialement, lors de la satisfaction de l'objectif de disponibilité minimum des données (SLO_{AV}).

(ii) Lorsque l'objectif de temps de réponse (SLO_{RT}) n'est pas satisfait. Dans ce cas, la création d'une nouvelle réplique est envisagée afin de satisfaire à nouveau le SLA. Ensuite, cette réplication doit être profitable pour le fournisseur.

La satisfaction de l'objectif de disponibilité minimum SLO_{AV} se fait au moment du stockage initial des relations, en créant un nombre minimum de répliques pour chaque relation. La satisfaction de l'objectif du temps de réponse (SLO_{RT}) est vérifiée en estimant le temps de réponse (RT_Q) du plan d'exécution de *Q*, fourni par l'optimiseur. La satisfaction de SLO_{RT} dépend alors de la valeur de RT_Q.

Une fois qu'un placement est trouvé pour la nouvelle réplique de telle sorte que l'objectif SLO_{RT} est satisfait à nouveau, le profit du fournisseur doit également être estimé. En effet, une nouvelle réplique est réellement créée uniquement si cette réplication est profitable pour le fournisseur. La rentabilité du fournisseur constitue alors un objectif à atteindre. Ainsi, le profit estimé du fournisseur pour l'exécution d'une requête (*Q_Profit*) doit être positif lorsqu'une réplication est envisagée. Les dépenses du fournisseur liées à la réplication tels que le coût économique de la réplication ou encore le coût des pénalités éventuelles payées aux locataires, doivent être prises en compte lors de cette estimation. En résumé, la décision de réplication repose sur : (a) un modèle de coûts qui estime RT_Q et (b) un modèle de coûts économique qui estime *Q_Profit* si une réplication est envisagée, avant l'exécution de *Q*.

Abréviation des paramètres utilisés	Signification
N_{ijk} *ou* MV_{ijk}	Machine Virtuelle $\in CD_{ij} \in RG_i$
CDij	Centre de Données $CD_{ij} \in RG_i$
RGi	Région RG_i
SLO_{AV}	Objectif de disponibilité minimum
SLO_{RT}	Objectif de temps de réponse
RT_Q	Temps de réponse estimé d'une requête *Q*
RT_SLO_PQ	Seuil critique de temps de réponse (une réplication par requête est considérée si $RT_Q > RT\_SLO\_PQ$)
RT_SLO_PSQ	Seuil de temps de réponse convenu dans le SLA (réplication par groupe de requêtes considérée si $RT_Q \in [RT\_SLO\_PSQ, RT\_SLO\_PQ]$ pour la $NSetQ^{ème}$ fois)
O_q	Opérateur $\subseteq$ requête *Q* d'un locataire
NodeP	Nœud sélectionné pour le placement d'une nouvelle réplique
$NB_{ijk.NodeP}$	Bande passante disponible entre N_{ijk} et *NodeP*
$Load_{ijk}$	Charge de travail estimée dans N_{ijk}
Repl_Fact_Ri	Facteur de réplication pour une relation *Ri*
Repl_Fact_Min	Facteur de réplication minimum pour la satisfaction de SLO_{AV}
PEQ	Plan d'exécution fourni pour *Q*
NEQ	Nœuds $\in RG_i$ exécutant *Q* et hébergeant les relations requises par *Q* et leurs répliques existantes
PF	Période de facturation
Q_Profit	Profit monétaire estimé pour l'exécution de *Q*
Q_Revenues	Revenus monétaires estimés pour l'exécution de *Q*
Q_Expenses	Dépenses monétaires estimées pour l'exécution de *Q*
RTE_Q	Temps de réponse effectif de *Q*
T_i	Locataire T_i
Ti_Amount	Montant monétaire payé par T_i au fournisseur pour l'exécution d'un nombre #*Q* de requêtes
Pen_RT	Montant monétaire de la pénalité payée par le fournisseur à T_i si $RTE_Q > RT\_SLO\_PSQ$

TABLE 4.1 - Abréviations des principaux paramètres utilisés.

Dans ce qui suit, nous détaillons les conditions nécessaires pour la création d'une nouvelle réplique lors de la satisfaction des objectifs SLO_{AV} et SLO_{RT} comme mentionné dans l'Algorithme 4.1.

4.3.1.1 Réplication pour satisfaire l'objectif SLO_{AV}

Les auteurs de (Wei et al., 2010) ont affirmé qu'un nombre élevé de répliques n'augmente pas forcément la disponibilité de données tandis que les coûts de la réplication augmentent en fonction du facteur de réplication. Ils ont aussi démontré qu'assurer un objectif de

disponibilité donné correspond au maintien d'un nombre minimum de répliques pour chaque relation R_i. Ceci est d'ailleurs adopté dans le système HDFS à travers la création de 3 répliques pour chaque unité de réplication. On parle alors de triplication (Cheng et al., 2012).

Comme indiqué dans l'Algorithme 4.1 (ligne 2), nous proposons de répliquer, initialement, chaque relation R_i (*Repl_Fact_Min*) fois afin de satisfaire une disponibilité minimum pour R_i. *Repl_Fact_Min* correspond alors au nombre minimum de répliques permettant de satisfaire SLO_{AV}. Ainsi, $Repl\_Fact\_R_i$, le nombre de répliques de R_i, doit toujours être supérieur à *Repl_Fact_Min*.

Concernant le placement de ces répliques, la satisfaction de l'objectif SLO_{AV} pour une relation $R_i \in N_{ijk}$ (avec $N_{ijk} \in CD_{ij} \in RG_i$) consiste à répliquer R_i initialement sur : (a) un nœud $N_{ijk'} \in CD_{ij}$ ($k' \neq k$), avec N_{ijk} étant le nœud le moins occupé $\in CD_{ij}$, (b) un nœud $\in CD_{ij'} \in RG_i$ (avec $j' \neq j$) avec $CD_{ij'}$ le CD voisin de CD_{ij}, et (c) un nœud dans chaque région $RG_{i'}$ ($i' \neq i$). En répliquant R_i dans toutes les régions, nous pourrons garantir un certain niveau de tolérance aux pannes. Cependant, la tolérance aux pannes n'est pas l'objet de nos travaux de recherche.

Algorithme 4.1 - Création de répliques.

Entrées : *Q, RT_Q, RT_SLO_PSQ, RT_SLO_PQ, NSetQ, Repl_Fact_Min, $Repl\_Fact\_R_i$. Initialement, NQ=0.*
Sortie : *Création éventuelle d'une nouvelle réplique sur NodeP.*

1. ***Begin***
   // Satisfaction initiale de l'objectif SLO_{AV}
2. ***While*** *($\exists R_i$ / $Repl\_Fact\_R_i$ < Repl_Fact_Min)* ***{****Création d'une réplique de R_i****}***
   // Satisfaction de l'objectif SLO_{RT}
3. ***Before*** *chaque exécution d'une requête Q*
4. ***{ If*** *(RT_Q < RT_SLO_PSQ)* ***then {****Pas de création de réplique****};***
5. ***Else { If*** *(RT_Q < RT_SLO_PQ)* ***then***
6. ***{if (****NQ < NSetQ)* ***then {****Pas de création de réplique; NQ++****}***
7. ***else*** *// **réplication par groupe de requêtes***
8. ***if*** *(NodeP est trouvé / (RT_Q < RT_SLO_PSQ and Q_Profit > 0)*
   then {*Création d'une réplique sur NodeP; NQ=0****}}***
9. ***Else*** *// RT_Q > RT_SLO_PQ // **réplication par requête***
10. ***if*** *(NodeP est trouvé / (RT_Q < RT_SLO_PSQ and Q_Profit > 0)*
    then {*Création d'une réplique sur NodeP; NQ=0****}***
11. ***}***
12. ***}***
13. *Exécution de Q*
14. ***End***

4.3.1.2 Réplication pour satisfaire l'objectif SLO_{RT}

Avant l'exécution de chaque requête *Q*, l'optimiseur du SGBD fourni un Plan d'Exécution de *Q* (*PEQ*) à notre stratégie. A ce moment, il est important de savoir si *PEQ* satisfait SLO_{RT} ou pas. Cela passe par l'estimation de son temps de réponse RT_Q. Il correspond au temps

estimé entre le début et la fin du traitement de Q (Ozsu and Valduriez, 2020). L'estimation du RT_Q est évoquée en détail dans la section 4.4. La valeur de RT_Q détermine les cas suivants :

(i) Il n'y a pas de réplication, c'est-à-dire qu'aucune réplique n'est créée, ce qui signifie que SLO_{RT} est satisfait. Cela se produit si $RT_Q < RT\_SLO\_PSQ$ (ligne 4 dans l'Algorithme 4.1). RT_SLO_PSQ est un seuil de temps de réponse convenu dans le SLA. Sa valeur est préalablement négociée entre le fournisseur et ses locataires (Yin et al., 2018). Cela dépend des besoins des applications. Ainsi, le locataire pourra payer plus cher pour un seuil de temps de réponse réduit, ce qui nécessite l'allocation de ressources importantes par le fournisseur. On parle alors d'un seuil de temps de réponse strict. En revanche, le locataire payera un montant plus ou moins réduit pour un seuil de temps de réponse élevé. Cela nécessite l'allocation de moins de ressources par le fournisseur. On parle alors d'un seuil de temps de réponse souple.

(ii) Une réplication peut être envisagée. Dans ce cas, deux types de réplication sont possibles : une *réplication par requête* ou une *réplication par groupe de requêtes*.

Réplication par requête. Si le temps de réponse estimé RT_Q est supérieur à RT_SLO_PQ, une réplication de données est immédiatement considérée (ligne 9 dans l'Algorithme 4.1). RT_SLO_PQ est un seuil critique de temps de réponse, avec $RT\_SLO\_PQ > RT\_SLO\_PSQ$. Il est préalablement établi par le fournisseur afin de limiter les pénalités payées aux locataires. Cette réplication est dite '*réplication par requête*' puisqu'elle est immédiatement considérée lorsque le temps de réponse estimé pour une requête unique dépasse RT_SLO_PQ.

Réplication par groupe de requêtes. Une création de réplique par requête peut augmenter le coût de la réplication d'une manière significative, notamment avec un taux élevé d'arrivée des requêtes. Afin de réduire le coût de réplication généré par des réplications répétitives, nous proposons de répliquer le plus souvent, par groupe de requêtes (Mokadem and Hameurlain, 2020). Dans ce cas, une réplication de données est considérée si le temps de réponse estimé pour une requête RT_Q appartient à l'intervalle [RT_SLO_PSQ, RT_SLO_PQ] pour un certain nombre de fois. C'est-à-dire qu'un certain nombre de requêtes antérieures étaient également dans cet intervalle. De cette manière, nous évitons de créer une nouvelle réplique à chaque fois que RT_Q dépasse un seuil RT_SLO_PSQ dit 'non critique'. Cela permet d'avoir davantage de réplications de type '*par groupe de requêtes*'. Dans l'Algorithme 4.1, une réplication *par groupe de requêtes* est considérée si RT_Q est dans [RT_SLO_PSQ, RT_SLO_PQ] pour la $NSetQ^{ème}$ fois (ligne 7). Dans cet algorithme, NQ correspond au nombre de requêtes pour lesquelles le temps de réponse estimé RT_Q appartient à cet intervalle

En résumé, l'objectif SLO_{RT} est satisfait si : (i) RT_Q est inférieur à RT_SLO_PSQ (ligne 4 dans l'Algorithme 4.1) ou (ii) pas plus de $NSetQ$ requêtes ont RT_Q dans l'intervalle [RT_SLO_PSQ, RT_SLO_PQ] (ligne 6 dans l'Algorithme 4.1). Dans ce cas, aucune réplique n'est créée. D'un autre coté, SLO_{RT} n'est pas satisfait si : (i) RT_Q est supérieur à RT_SLO_PQ (*réplication par requête*) ou (ii) RT_Q est dans [RT_SLO_PSQ, RT_SLO_PQ] pour la $NSetQ^{ème}$ fois (*réplication par groupe de requêtes*).

4.3.1.3 Prise en compte de la profitabilité du fournisseur

En contrepartie de l'exécution des requêtes des locataires, les fournisseurs de Cloud perçoivent des revenus suite à l'allocation des ressources nécessaires à l'exécution de ces requêtes. Ces revenus correspondent au montant facturé aux locataires durant une Période de Facturation (PF). Ils dépendent de plusieurs paramètres tels que la taille des données traitées, le seuil de temps de réponse ou encore le nombre maximum de requêtes à traiter. Nous rappelons que l'ensemble de ces paramètres ainsi que le montant des loyers sont définis dans le SLA.

Lorsqu'une création d'une réplique est considérée dans le but de la satisfaction à nouveau du SLO_{RT} (par requête ou par groupe de requêtes), cette réplique n'est réellement créée que si un nœud susceptible de recevoir une nouvelle réplique est trouvé, de sorte que $RT_Q < RT\_SLO\_PSQ$. De plus, cette réplication doit être rentable pour le fournisseur.

Assurer la rentabilité du fournisseur lors de l'exécution de Q signifie que le profit économique estimé du fournisseur (*Q_Profit*) pour l'exécution de Q doit être positif lorsqu'une réplication est envisagée (lignes 8 et 10 dans l'Algorithme 4.1). Le montant estimé de ses revenus (*Q_Revenues*) doit alors être supérieur au montant estimé de ses dépenses (*Q_Expenses*) pour l'exécution de Q tout en approvisionnant d'autres locataires. L'estimation de ces coûts économiques, indiqués dans la Formule (4.1), est examinée dans la section relative au modèle de coûts économique.

$$Q\_Profit = Q\_Revenues - Q\_Expenses \tag{4.1}$$

4.3.2 Placement et choix des données à répliquer

Il a été prouvé que le placement des répliques dans un système à grande échelle est un problème NP-difficile (Kumar et al., 2014). Sans un placement efficace de ces répliques, celles-ci peuvent être distribuées de manière déséquilibrée. En conséquence, certains nœuds peuvent contenir plus de répliques qu'ils peuvent en supporter, ce qui génère une surcharge au niveau de ces nœuds.

Soit PEQ : $<Q, NEQ>$ un plan d'exécution fourni par le SGBD pour l'exécution de Q : $\{O_0, O_2, ..., O_q, ... O_{n-1}\}$ constituée d'un ensemble de n opérateurs, par exemple des opérateurs de jointure. Ces opérateurs doivent être exécutés dans des $CD_{ij} \in RG_i$. Ici, NEQ : $\{N_{ijk}, N_{ijk'}, ..., N_{ij'k}\}$ avec $(j \neq j' \ \& \ k \neq k')$ correspond à l'ensemble des nœuds exécutant les opérateurs de Q ainsi que les nœuds stockant toutes les relations requises par Q et leurs répliques déjà existantes. Rappelons que dans notre cas, ces nœuds sont situés dans la même région RG_i.

Supposons maintenant que PEQ ne satisfait pas l'objectif SLO_{RT}. Une réplication de données est alors envisagée afin de satisfaire à nouveau SLO_{RT} tout en prenant en compte la rentabilité du fournisseur. Cela passe par un placement stratégique de toute nouvelle réplique. Il faudrait alors sélectionner un nœud de placement *NodeP* qui pourrait recevoir cette nouvelle réplique afin de satisfaire SLO_{RT} de manière rentable. Bien évidemment, *NodeP* doit avoir une charge faible, un espace de stockage suffisant et une bande passante suffisante permettant d'accéder à la nouvelle réplique.

Avant tout, il est important de déterminer quel opérateur O_q a causé la violation du SLA (ici, la non satisfaction de SLO_{RT}). Cela peut être dû à une défaillance de ressources, par exemple en termes de bande passante du réseau ou de capacité de calcul (CPU). Dans ces

cas, répliquer les relations associées à cet opérateur avant même l'exécution de *Q* peut permettre la satisfaction du SLA.

L'identification du meilleur nœud de placement nécessite la visite de tous les nœuds du système, ce qui ne permet pas forcément la satisfaction du SLA. En effet, si tous les nœuds du Cloud doivent être évalués en termes de charge de travail, de stockage et de ressources réseau, cela engendrerai un temps de réponse élevé, ce qui impacterai le passage à l'échelle dans un tel système.

Par ailleurs, les systèmes Cloud peuvent être hétérogènes à bien des égards. Les coûts de stockage, de processeur et de réseau peuvent varier d'un CD à l'autre pour plusieurs raisons, par exemple la différence de prix des matières premières. En outre, tous les CDs ne peuvent pas satisfaire le temps de réponse des requêtes provenant des locataires d'une région donnée. Accéder à une relation située sur un CD à Toulouse peut satisfaire l'objectif SLO_{RT} contrairement au CD situé à Paris, alors que seuls certains CDs en Chine, avec des prix de ressources réduits, peuvent satisfaire SLO_{RT}. Il est alors difficile de trouver le meilleur emplacement qui satisfait SLO_{RT} au coût le plus bas.

Heuristique 4.1 - Placement de répliques.

Entrées : *PEQ, NEQ,* $N_{ijk} \in NEQ \in RG_i$, $O_q \subseteq Q$ *qui demande* R_r *(relation distante* $\in NodeR$*) et/ou* R_l *(*$\in NodeO$ *exécutant* O_q*). Initialement,* **SLO_{RT} n'est pas satisfait** et *NEQ'=NEQ.*
Sortie : *Sélection éventuelle de NodeP (nœud de placement de la réplique créée).*

Begin

// SLO_{RT} non satisfait en raison d'un transfert de R_r via une faible bande passante
If *($R_r \neq \emptyset$)* ***then***
{Trouver NodeT $\in NEQ$ *qui demande* R_r*/* $NB_{NodeT.NodeR} = MIN_{jk}(NB_{ijk.NodeR})$
Trouver NodeP $\in CD_{ij}$*/ NodeP* $\neq$ *NodeR and* $NB_{NodeT.NodeP} = MAX_{jk}(NB_{NodeT.ijk})$
if *((NodeP* $\neq \emptyset$*) and S(NodeP) >* S_{Rr} *and* $(RT_Q < RT\_SLO\_PSQ)$*)*
then *{NEQ <-- NEQ + NodeP; goto 12}*
else *{NEQ <--NEQ - NodeT;* ***if*** *NEQ=*$\emptyset$ ***then*** *{NEQ = NEQ'; goto 14}* ***else*** *goto 3}*
}
Else *// SLO_{RT} non satisfait en raison d'un nœud NodeO surchargé*
{Trouver NodeO $\in NEQ$ *qui demande* R_l */* $Load_{NodeO} = MAX_{jk}(Load_{ijk})$
Trouver NodeP $\in CD_{ij}$*/ NodeP* $\neq$ *NodeO* and $Load_{NodeP} = MIN_{jk}(Load_{ijk})$
if *((NodeP* $\neq \emptyset$*) and S(NodeP) >* S_{Rl} *and* $(RT_Q < RT\_SLO\_PSQ)$*)*
then *{NEQ <-- NEQ + NodeP; goto 12}*
else *{NEQ <-- NEQ - NodeO;* ***if*** *NEQ=*$\emptyset$ ***then*** *{NEQ = NEQ'; goto 14}* ***else*** *goto 8}*
}
// Estimation du profit du fournisseur
If *(Q_Profit > 0)* ***then***
R_r *(R_l respect.) est répliquée de NodeR (NodeO respect.) vers NodeP*
Exécution de Q
End

Dans la stratégie PEPR (Tos et al., 2018), *NodeP* peut être sélectionné en dehors de RG_i. PEPR permet de sélectionner un CD offrant les ressources les moins chères puis, de trouver un

nœud qui satisfait l'objectif SLO_{RT}. Certes, cela réduit les coûts du fournisseur. Néanmoins, cela nécessite la visite de tous les CDs dans toutes les régions, ce qui rallonge le temps de réponse. Compte tenu du fait qu'un système Cloud consiste en un certains nombre de CDs contenant un nombre en apparence infini de nœuds, il est préférable de trouver un nœud de placement acceptable, pas forcément au prix le plus bas, ce qui permet de réduire l'espace de recherche. Dans (Mokadem and Hameurlain, 2020), nous avons proposé une heuristique de placement d'une réplique permettant de trouver un emplacement acceptable qui satisfait l'objectif SLO_{RT} pour le locataire tout en prenant en compte le profit du fournisseur. L'Heuristique 4.1 réduit sensiblement l'espace de recherche en éliminant les régions inappropriées, puis en recherchant le premier nœud approprié au lieu de rechercher le meilleur. En effet, compte tenu du nombre de CDs géographiquement distribués sur le Cloud, seul un certain nombre de ces CDs disposent d'une largeur suffisante de la bande passante. Le nœud de placement *NodeP* pouvant contenir la nouvelle réplique est alors sélectionné parmi les nœuds des CDs appartenant à RG_i recevant Q.

Soient les dénotations suivantes : S_{Rr} correspond à la taille d'une relation distante R_r requise par $O_q \subseteq Q$, S_{Rl} correspond à la taille d'une relation R_l requise par O_q exécuté sur *NodeO* (R_l est une relation $\in$ *NodeO*), *S(NodeP)* est la capacité de stockage disponible sur *NodeP*, $Load_{ijk}$ correspond à l'estimation de la charge de travail sur N_{ijk} fourni par le gestionnaire de charge de travail dans CD_{ij}, $NB_{NodeR.ijk}$ est la bande passante disponible entre *NodeR et* N_{ijk}, et *Q_Profit* correspond à l'estimation de profit du fournisseur lorsqu'une requête Q est exécutée en considérant les coûts de la création éventuelle d'une réplique sur *NodeP*.

Le placement d'une nouvelle réplique commence par trouver la ressource défaillante à l'origine de la non satisfaction de l'objectif SLO_{RT}. Un goulet d'étranglement peut être généré par : (i) le transfert d'une relation distante R_r ($\in$ *NodeR*) via une faible bande passante lors du traitement de O_q. Dans ce cas, satisfaire SLO_{RT} consiste à trouver *NodeP* qui pourra recevoir une réplique de R_r tels que *NodeP* a une meilleure bande passante avec le nœud qui accède à R_r ou, (ii) la surcharge d'un nœud qui exécute O_q alors que seules des relations locales R_l sont requises sur ce nœud. Dans ce cas, O_q doit être exécuté sur un autre nœud moins chargé *NodeP* qui contiendra également une nouvelle réplique de R_l. Ci-dessous, la sélection de *NodeP* est détaillée pour chaque scénario.

4.3.2.1 Scénario 1 : SLO_{RT} non satisfait en raison d'un transfert de données via une faible bande passante du réseau

Cette situation se produit lorsque O_q accède à une relation R_r disponible sur *NodeR* distant et possédant une faible bande passante avec le nœud exécutant l'opérateur O_q (ligne 2 de l'Heuristique 4.1). Nous commençons par trouver le nœud *NodeT* $\in$ *NEQ* dans CD_{ij}, disposant de la plus faible bande passante pour rapatrier $R_r \in$ *NodeR* (ligne 3), c'est-à-dire, $NB_{NodeT.NodeR} = MIN_{jk}(NB_{ijk}.NodeR)$ avec $N_{ijk} \in$ *NEQ*. Ensuite, *NodeP* doit être trouvé afin de recevoir une nouvelle réplique de R_r. La bande passante est vérifiée entre chaque $N_{ijk} \in CD_{ij}$ ($N_{ijk} \neq$ *NodeR*) et *NodeT*. Le nœud dans CD_{ij} ayant la meilleure bande passante vers *NodeT* est choisi pour être *NodeP*, c'est-à-dire $NB_{NodeT.NodeP} = MAX_{jk}(NB_{NodeT.ijk})$ (ligne 4). De plus, *NodeP* doit avoir assez de ressource de stockage pour contenir une réplique de R_r.

Si *NodeP* est trouvé tel que (i) l'objectif SLO_{RT} est satisfait ($RT_Q < RT\_SLO\_PSQ$) (ligne 5) et ensuite (ii) si le fournisseur estime qu'il réalise un bénéfice en considérant cette réplication (ligne 12), alors d_r est réellement répliqué sur *NodeP* (ligne 13). Finalement, Q est exécutée et

dorénavant, *NEQ* contient également *NodeP*. Dans le cas contraire, le processus est répété avec d'autres relations distantes requises par d'autres nœuds $\in NEQ$ (ligne 6).

4.3.2.2 Scénario 2 : SLO_{RT} non satisfait en raison d'un nœud surchargé

Cette situation se produit si SLO_{RT} n'est pas satisfait car certains nœuds sont surchargés (ligne 7 dans l'Heuristique 4.1). Nous commençons alors par trouver le nœud le plus chargé (le plus occupé) dans *NEQ* (ligne 8), c'est-à-dire $Load_{NodeO} = MAX_{jk}(Load_{ijk})$ avec $N_{ijk} \in NEQ$. Supposons que ce nœud est $NodeO \in CD_{ij}$. Supposons maintenant qu'une relation R_l se trouve sur *NodeO* et participe à l'exécution de l'opérateur O_q. La recherche d'un nœud de placement consiste alors à trouver un nœud moins chargé $\in CD_{ij}$ avec $NodeP \neq NodeO$ (ligne 9), c'est-à-dire $Load_{NodeP} = MIN_{jk}(Load_{ijk})$, de sorte qu'il puisse recevoir une réplique de R_l, c'est-à-dire S $(NodeP) > S_{Rl}$. Il doit également exécuter O_q tels que $RT_Q < RT\_SLO\_PSQ$ (ligne 10). Si SLO_{RT} n'est pas encore satisfait, le processus est répété avec les autres nœuds $\in NEQ$ (ligne 11). Par contre, si *NodeP* est trouvé de telle sorte que SLO_{RT} soit satisfait et que le fournisseur présente un avantage économique, une réplique de R_l est créée réellement sur *NodeP*. Enfin, *Q* est exécutée et dorénavant, *NEQ* contient également *NodeP*.

Dans les deux scénarios, si aucun *NodeP* n'est trouvé ou si la réplication envisagée ne peut être rentable, *Q* est exécutée sur l'ensemble *NEQ* initial et des pénalités sont payées par le fournisseur à son locataire.

4.3.3 Ajustement du nombre de répliques

Déterminer le nombre de répliques d'une relation est une tâche importante pour toute stratégie de réplication de données. Très peu de répliques ne permettent pas de satisfaire le niveau de performance attendu, tandis que la création de beaucoup de répliques peut entraîner un gaspillage de ressources. En effet, un sur-approvisionnement statique de répliques constitue une solution naïve qui peut entraîner une sur-utilisation des ressources et une baisse des revenus pour le fournisseur (Bonvin et al., 2011).

Certaines stratégies de réplication de données (Fu et al., 2013) prédéfinissent un facteur de réplication pour un système de gestion de données. Ensuite, ce nombre de répliques peut être réajusté périodiquement. Dans la stratégie proposée dans (Tos et al., 2018), il est inutile de créer plusieurs répliques à la fois pour une relation. Une nouvelle réplique peut être créée à chaque fois qu'il est nécessaire de satisfaire le SLA à condition que cette réplication soit rentable, c'est-à-dire que des répliques sont créées de manière dynamique et il n'y a pas de limite supérieure pour le nombre de répliques d'une relation donnée. Il est également probable que plusieurs relations puissent causer un goulet d'étranglement et donc, une violation du SLA. Dans ce cas, on peut envisager de répliquer autant de relations qu'il est nécessaire jusqu'à la satisfaction du SLO_{RT}, ce qui évite le payement de pénalités par le fournisseur au locataire. Toutes les répliques sont évaluées périodiquement pour déterminer leur éligibilité à être retirées du système en fonction de la satisfaction du SLO_{RT}. La durée d'une telle période est déterminée par le fournisseur. Dans la solution proposée dans (Mokadem and Hameurlain, 2020), nous avons également opté pour une réplication incrémentale. Puis, une réplique est maintenue ou retirée suivant la satisfaction des objectifs SLO_{RT} et SLO_{AV}.

La création d'une nouvelle réplique engendre une consommation de ressources, par exemple en termes de bande passante et de stockage, ce qui augmente les dépenses du

fournisseur. Le fournisseur doit alors éviter de créer inutilement des nouvelles répliques. Certaines répliques peuvent aussi devenir inutilisées au fil du temps en raison de l'évolution de la demande des locataires à travers des accès de moins en moins fréquents à ces répliques. Le fait de conserver ces répliques dans le système consomme alors un stockage précieux et entraîne des coûts inutiles pour le fournisseur (Liu et al., 2018). Dans la stratégie proposée dans (Cheng et al., 2012) par exemple, les répliques de données impopulaires sont effacées afin de réduire la consommation de ressources de stockage tout en garantissant une certaine tolérance aux pannes.

Nous avons proposé un algorithme d'ajustement dynamique de ces répliques (Mokadem and Hameurlain, 2020). Lorsque la charge de travail des nœuds est réduite, les répliques inutiles doivent être supprimées. Cela réduit la consommation de ressources pour le fournisseur, ce qui augmente ses bénéfices. Le nombre de répliques est ajusté au fur et à mesure du traitement des requêtes. Cet ajustement se fait en fonction de l'estimation du temps de réponse comme indiqué dans l'Algorithme 4.2. Il se produit lorsque SLO_{RT} est largement satisfait. Cette situation intervient lorsque RT_Q est très inférieur à RT_SLO_PSQ, c'est-à-dire que $RT_Q \leq \beta \times RT\_SLO\_PSQ$, avec $0 < \beta < 1$. La valeur de β est établie par le fournisseur. Cela signifie que certains nœuds sont sous-chargés. Certaines répliques sont alors moins consultées et donc inutiles.

Algorithme 4.2 - Suppression de répliques inutiles.

Entrées : *Q, NEQ, RT_Q, RT_SLO_PSQ, β, Repl_Fact_Min, une période T.*
Sorties : *R_i (relation la moins populaire requise par Q), N_Remov (nœud ∈ NEQ contenant la réplique la moins accédée de R_i).*

1. Begin
2. If *($RT_Q \leq \beta \times RT\_SLO\_PSQ$)* ***then***
3. *{Trouver une relation $R_i \in NEQ$ / R_i est la relation (requise pas Q) la moins populaire*
4. ***If*** *(Repl_Fact_ R_i > Repl_Fact_Min)* ***then***
5. *{Compresser la réplique la moins accédée de $R_i \in N\_Remov$ (supprimée après T);*
6. *Repl_Fact_ R_i = Repl_Fact_ R_i -1}*
7. ***else*** *goto 3 et répéter avec une autre relation requise par Q}*
8. End

La popularité des données constitue un paramètre important que de nombreuses stratégies exploitent en répliquant les données les plus demandées (Ananthanarayanan et al., 2011). La popularité d'un fragment R_i est déterminée en analysant l'accès des utilisateurs à R_i pendant une unité de temps (Ranganathan et al., 2002).

Concernant la réplique inutile, nous sélectionnons la réplique de la relation la moins populaire, requise par *Q*. Supposons que cette relation est R_i. Alors, la réplique la moins demandée (la moins populaire) de R_i doit être trouvée dans *NEQ* afin qu'elle soit inactive. Supposons que cette réplique de R_i est hébergée sur un nœud $N\_Remov \in NEQ$ (ligne 3 dans l'Algorithme 4.2). Elle doit alors être retirée de la liste des répliques disponibles de R_i. Dans notre proposition, nous avons choisi de ne pas la supprimer immédiatement car elle pourrait être utilisée à nouveau dans un proche avenir. Une solution consiste à compresser cette réplique afin de réduire l'espace de stockage utilisé (Liu et al., 2018). Cela permet d'éviter la recréation de cette réplique si elle est nécessaire dans un avenir proche. Néanmoins, si cette

réplique n'est pas accédée durant une période donnée T (définie par le fournisseur), elle est définitivement supprimée (ligne 5).

D'un autre coté, assurer la disponibilité des données est un autre objectif que le fournisseur doit aussi satisfaire (Stamou et al., 2013). Par conséquent, l'objectif de disponibilité minimum SLO_{AV} ne doit pas être ignoré lors de l'ajustement du nombre de répliques. Le nombre de répliques ne peut alors être inférieur au nombre de répliques minimum *Repl_Fact_Min* lors des opérations de suppression de répliques. De ce fait, la suppression d'une réplique donnée n'est pas systématiquement effectuée, car SLO_{AV} doit également être satisfait, c'est-à-dire que $(Repl\_Fact\_R_i \geq Repl\_Fact\_Min)$ doit être tout le temps vérifiée.

4.4 MODELE DE COUTS POUR L'EVALUATION DE REQUETES RELATIONNELLES

Le traitement d'une requête relationnelle dans un environnement distribué implique un ensemble d'étapes comprenant : (i) l'analyse (syntaxique et sémantique) de la requête permettant de vérifier si cette requête est exprimée correctement, (ii) la transformation de cette requête en une requête répartie équivalente exprimée sur les fragments et (iii) l'optimisation (logique et physique) de cette requête. Une requête relationnelle, sous sa forme algébrique, est représentée par un graphe d'opérateurs s'appliquant sur un ensemble de relations/fragments de relations. L'optimisation de ce graphe (par l'optimiseur du SGBD) permet de transformer le graphe d'opérateurs en un plan d'exécution d'opérations de bas niveau pour l'accès à ces relations. L'optimisation logique permet la réduction de la taille des données manipulées en appliquant des règles de transformation des arbres algébriques. L'optimisation physique permet le choix des algorithmes d'exécution des opérateurs et leur ordre en se basant sur un modèle de coûts permettant le calcul d'un ensemble de métriques pour trouver un plan d'exécution séquentiel optimal ou proche de l'optimal. On notera par *PEQ*, ce plan d'exécution généré pour la requête *Q*.

Un SGBD classique implanté sur une architecture parallèle est appelé un SGBD parallèle. Parmi les principaux produits et prototypes de SGBD parallèles à mémoire distribuée, nous citons Teradata[20] et Oracle Parallel Query (Oracle, 2019). Lors de l'évaluation d'une requête dans un environnement d'exécution parallèle, le plan d'exécution séquentiel produit par l'optimiseur du SGBD est transformé en un plan d'exécution parallèle. Dans nos travaux de recherche, nous considérons que ce plan est fourni à notre stratégie de réplication. L'optimiseur fournit également l'emplacement de toutes les relations requises, leurs répliques existantes ainsi que des statistiques et estimations relatives à l'exécution de *PEQ* (Kabra and Dewitt, 1998).

Nous rappelons qu'une requête *Q* d'un locataire peut nécessiter plusieurs relations distribuées sur plusieurs nœuds, eux même répartis sur différents CDs $\in$ une même région RG_i. Le traitement d'une telle requête peut nécessiter l'exécution d'une multitude d'opérateurs tels que des jointures relationnelles. En d'autres termes, nous supposons que l'exécution de l'ensemble des opérateurs de *Q* se fait uniquement dans une région RG_i.

Dans ce qui suit, nous présentons notre modèle de coûts sur lequel est basée la prise de décision de création d'une réplique. Nous rappelons que cette création n'est envisagée que si l'objectif de temps de réponse SLO_{RT} n'est pas satisfait. Cela passe par l'estimation du temps de réponse de *PEQ* avant son exécution.

[20] http://www.teradatatech.com/?p=103 avril 2020

4.4.1 Estimation du temps de réponse

L'estimation du temps de réponse d'une requête sans opérations dépendantes, par exemple sans opérateurs de jointure, est une tâche relativement simple. L'estimation du temps de réponse n'est qu'une approximation des temps d'exécution des différents opérateurs et des différents transferts de données. Néanmoins, le résultat d'un opérateur peut souvent être l'entrée d'un autre opérateur en tant que résultat intermédiaire. L'estimation du temps de réponse d'une requête comportant des opérations dépendantes est alors plus difficile. La recherche des chaines de pipeline, la complexité de calcul de chaque opérateur dans ces chaines de pipeline et la taille des données (y compris celles expédiées à partir de nœuds distants) constituent des paramètres importants lors de l'estimation du temps de réponse d'une telle requête. Pour faire face à ces problèmes, l'estimation du temps de réponse des requêtes a fait l'objet de plusieurs travaux de recherche (Hsiao et al., 1994; Swami et Schiefer, 1994; Tomov et al., 2004; Vengerov et al., 2015). C'est pourquoi nous tirons profit de ces travaux pour proposer un modèle d'estimation du temps de réponse adapté à la stratégie de réplication de données que nous proposons.

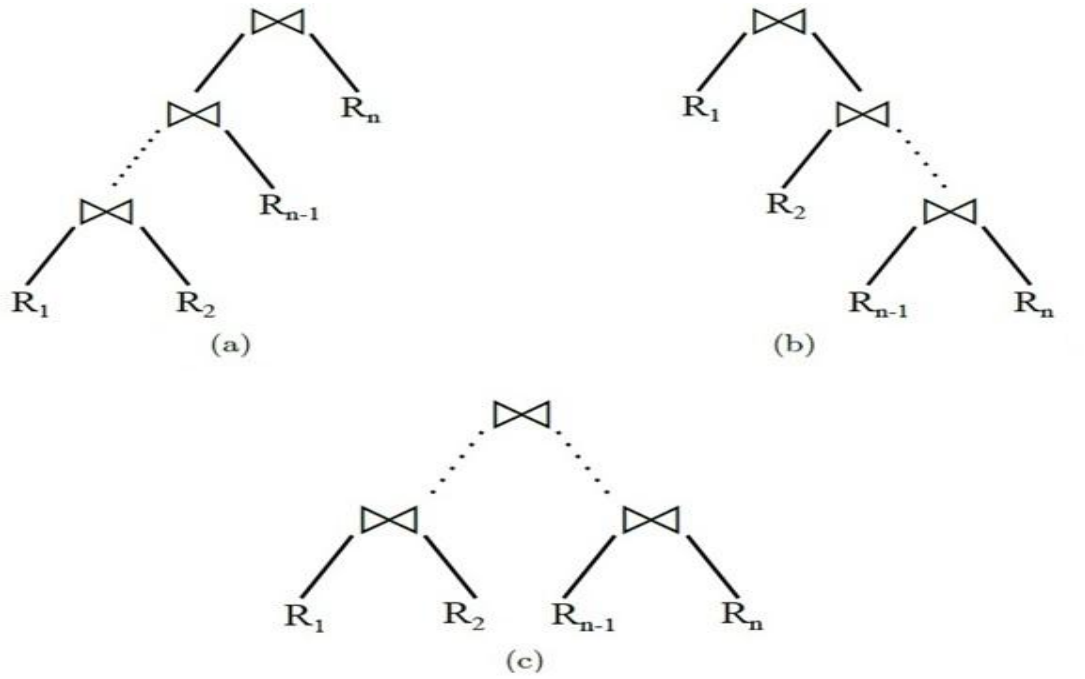

FIGURE 4.4 - Plans d'exécution d'une requête : (a) linéaire gauche, (b) linéaire droit et (c) ramifié (Bushy).

Nous nous intéressons aux requêtes composées de plusieurs opérateurs de jointure. Il existe plusieurs algorithmes pour l'exécution d'une jointure, par exemple par produit cartésien, par tri-fusion et par hachage. Le principe d'une jointure par hachage entre deux relations est de construire d'abord une table de hachage en mémoire, avec la plus petite des deux relations, via l'opérateur *Build*. Ensuite, le résultat de la jointure est produit via l'opérateur *Probe* (Hameurlain et al., 1996; Ozsu and Valduriez, 2020). Plusieurs variables de l'algorithme de jointure par hachage ont été proposés dans la littérature, par exemple la jointure par hachage simple, 'Grace hash join' et la jointure par hachage hybride. Dans ce qui suit, nous considérons l'algorithme de jointure par hachage simple.

Notre modèle d'estimation du temps de réponse couvre trois types de plans d'exécution : le plan linéaire gauche, le plan linéaire droit et le plan ramifié (Bushy) comme illustré dans la Figure 4.4. Nous rappelons que la génération de plans d'exécution de requêtes et la sélection d'un plan d'exécution approprié ne font pas l'objet de nos travaux de recherche.

Initialement, notre stratégies reçoit, pour chaque requête Q, un plan d'exécution PEQ : <Q, NEQ> qui sera exécutée sur un ensemble de nœuds $\in NEQ$. Notre stratégie commence par vérifier si le plan PEQ reçu satisfait bien l'objectif SLO_{RT}. Cela passe par l'estimation de RT_Q. qui tient compte de l''estimation de toutes les ressources nécessaires à l'exécution de PEQ :

(i) Les coûts de traitement de la requête (*CPU_Q_Cost*).

(ii) Les coûts d'accès aux données incluant les entrées/sorties (*IO_Q_Cost*).

(iii) Les coûts de transfert de données (*Transf_Q_Cost*) entre différents nœuds $\in NEQ$.

Cette estimation est une tâche complexe si l'on considère toutes les formes de parallélisme que nous décrivons plus loin. En effet, la consommation de ces ressources ne se fait pas de manière séquentielle. L'estimation de ces ressources se base alors inévitablement sur quelques simplifications afin d'estimer rapidement le temps de réponse des requêtes. Un modèle trop complexe augmenterait le temps nécessaire à cette estimation, ce qui annulerait les gains de performances attendus de cette réplication. À titre d'exemple, nous supposons que la CPU est occupée pendant les opérations d'E/S et que la consommation en terme de ressources CPU et d'E/S ne se produit pas simultanément sur un seul nœud. Nous bénéficions de plusieurs travaux dans les SGBD distribués, par exemple (Tomov et al., 2004), portant sur l'estimation du temps de réponse de requêtes relationnelles.

Par ailleurs, nous supposons qu'à tout moment, une ressource n'est utilisée que par une requête et qu'il n'y a aucune interférence avec d'autres requêtes afin d'avoir une estimation précise. Dans ce contexte, nous bénéficions de quelques travaux concernant l'isolation de ressources (Narasayya et al., 2013).

4.4.2 Exploitation du parallélisme

L'exploitation du parallélisme dans les systèmes distribués permet d'accélérer le traitement des requêtes des locataires. A partir des approches de placement de données et des méthodes de répartition, on distingue: (i) le parallélisme inter-requêtes lorsque plusieurs requêtes sont exécutées en parallèle par différents processeurs et (ii) le parallélisme intra-requête lorsque plusieurs processeurs coopèrent pour exécuter en parallèle les opérateurs d'une même requête. Il y a deux formes de parallélisme intra-requête : (a) le parallélisme intra-opération et (b) le parallélisme inter-opérations.

Dans ce qui suit, nous décrivons l'exploitation du parallélisme intra-requête pour des modèles d'exécution basés sur la fragmentation. D'abord, nous estimons le temps de réponse d'un opérateur relationnel lors de l'exploitation du parallélisme intra-opérateur. Puis, nous estimons le temps de réponse d'une requête en exploitant le parallélisme inter-opérateurs (indépendant et pipeline). Chaque processeur n'accède qu'à un sous ensemble des relations sur une architecture, rappelons le, à mémoire distribuée (Shared-Nothing) (Oszu and Valduriez, 2020). Puis, nous estimons le temps de réponse d'une requête exécutée sous chaque forme de parallélisme.

4.4.2.1 Parallélisme intra-operateur

Le parallélisme intra-opérateur concerne l'exécution en parallèle d'un opérateur relationnel O_q sur plusieurs nœuds, avec des données obtenues par une méthode de répartition. Il repose sur la décomposition de O_q en un ensemble de sous opérateurs indépendants $O_{q,k}$ généralement appelés instances de l'opérateur initial. Cette décomposition est rendue possible grâce à la fragmentation de chaque relation de base R en fragments (R_1, R_2, …, R_n). Chaque sous opérateur $O_{q,k}$ est exécuté sur un seul processeur et porte sur une partie des données, par exemple un fragment R_k de R. Il existe trois méthodes basiques de répartition (fragmentation) de données : circulaire ou Round-Robin (les relations sont réparties tuple par tuple dans l'ordre sur les nœuds), par hachage (une fonction de hachage est appliquée à chaque tuple pour déterminer le nœud associé) et par intervalle (les relations sont réparties avec des opérations de restriction par rapport à un attribut choisi par l'administrateur) (Oszu and Valduriez, 2020).

En fragmentant les opérandes d'un seul opérateur, il est possible de décomposer chaque opérateur en sous-opérateurs identiques conduisant à générer le parallélisme intra-opérateur. Certains opérateurs, comme la sélection et la projection, peuvent être facilement décomposés en tâches parallèles. Cette décomposition est plus complexe avec d'autres opérateurs tels que la jointure. Cela rend l'estimation du temps de réponse encore plus difficile.

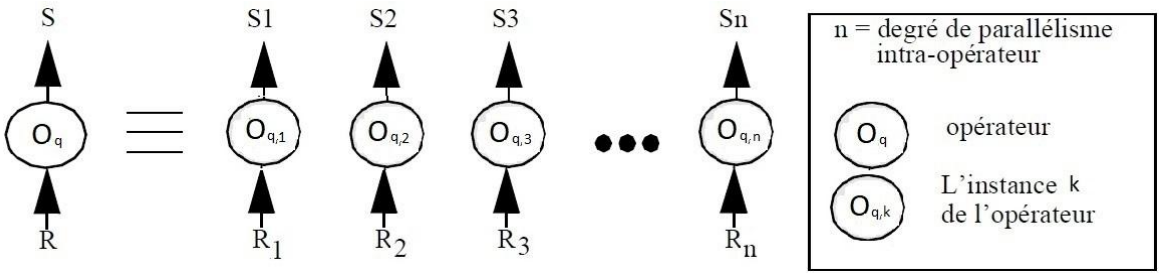

FIGURE 4.5 - Exécution en parallèle de l'opérateur Oq (parallélisme intra-opérateur).

Dans la Figure 4.5 (Bouganim, 1997), les sous opérateurs {$O_{q,1}$, $O_{q,2}$,…, $O_{q,n}$} portent sur n fragments {R_1, R_2, ..., R_n} disponibles sur plusieurs nœuds. L'exécution de O_q consiste à exécuter en parallèle tous ces sous-opérateurs qui produisent {$S1$, $S2$,…Sn}. Le résultat final est obtenu par la combinaison des résultats générés par chaque nœud.

$$RT_{Oq} = MAX_k\,(RT_{Oq,k}\,,\ T_{Tr},\ T_{Fin}) \mid O_{q,k} \in O_q \qquad (4.2)$$

Quelle que soit la manière dont la fragmentation a été effectuée par le SGBD, le temps de réponse d'un opérateur O_q exécuté sur plusieurs nœuds correspond au temps le plus long nécessaire à l'exécution de $O_{q,k}$ sur l'un de ces nœuds. Par ailleurs, les résultats {$S1$, $S2$,…Sn} doivent être transférés vers une destination finale pour produire le résultat. Comme indiqué dans la Formule (4.2), l'estimation de RT_{Oq} (temps de réponse de O_q) tient compte alors du transfert des résultats intermédiaires (T_{Tr}) et de la production du résultat (T_{Fin}), par exemple le temps nécessaire à une opération d'union pour joindre des relations fragmentées horizontalement.

4.4.2.2 Parallélisme inter-opérateurs (indépendant et pipeline)

Le parallélisme inter-opérateurs consiste à exécuter en parallèle plusieurs opérateurs d'une même requête *Q* ayant pour plan d'exécution *PEQ*. Il existe deux types de parallélisme inter-opérateurs :

(a) Le parallélisme inter-opérateurs indépendant qui consiste à exécuter en parallèle deux opérateurs indépendants du graphe de la requête. Les plans d'exécution de type arbre ramifié, souvent appelé 'Bushy', offrent l'opportunité de mettre en ouvre cette forme de parallélisme. La Figure 4.6 (a) montre l'exploitation du parallélisme inter-opérateurs indépendant pour l'exécution des opérateurs O_1 et O_2.

(b) Le parallélisme inter-opérateur dépendant, appelé aussi parallélisme *pipeline*, correspond à une exécution parallèle des opérations qui communiquent. Il consiste à exécuter deux opérateurs ayant un lien producteur-consommateur. Le résultat du producteur (O_1 sur la Figure 4.6 (b)) n'est pas matérialisé mais est considéré comme une relation intermédiaire *T* qui sera envoyé en pipeline à l'opérateur O_2 suivant le même principe qu'un pipe unix. Cela évite la matérialisation inutile de la relation T. On appelle une chaine pipeline l'ensemble des opérateurs O_1 et O_2 exécutés en pipeline sur la Figure 4.6 (b).

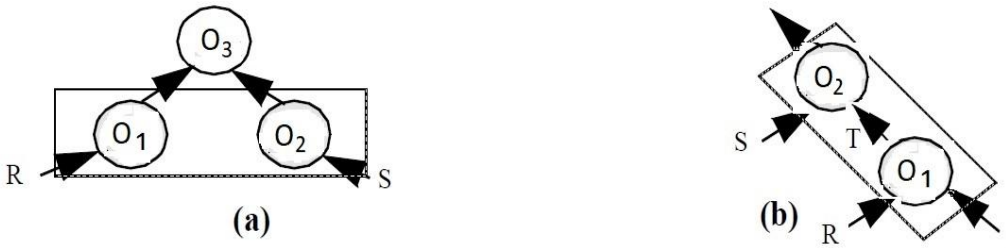

FIGURE 4.6 - Parallélisme inter-opérateurs : (a) indépendant (b) et pipeline.

Un plan d'exécution *PEQ* peut comporter une ou plusieurs chaines de pipeline. Afin de déterminer les chaînes de pipeline dans un *PEQ*, on évalue dans un premier temps dans quel ordre les opérateurs vont être exécutés. Lorsque nous traitons le parallélisme entre opérateurs, nous prenons en compte à la fois l'exécution indépendante et l'exécution en pipeline. Lors de l'exécution en pipeline, certains opérateurs bloquent le flux de données entre opérateurs (Garofalakis and Ioannidis, 1996; Wu et al., 2013). En effet, les tables de hachage doivent être générées avant l'étape de sondage (Probe) lors de la jointure par hachage. En conséquence, ces opérateurs de blocage sont identifiés et la requête est décomposée en chaînes de pipeline (Tomov et al., 2004).

La Figure 4.7 (Tos, 2017) illustre la manière dont les chaînes de pipeline sont identifiées dans les plans de requête linéaire gauche, linéaire droit et ramifié. L'ensemble des opérateurs appartenant à chaque chaîne de pipeline sont également exécutés dans un pipeline, jusqu'à ce qu'il soit le dernier opérateur de la chaîne concernée.

Comme un plan d'exécution *PEQ* peut contenir n'importe quel nombre de chaînes de pipeline et de dépendances entre elles, il est possible de modéliser cela sous la forme d'un arbre $Y <V, E>$. L'ensemble *V* représente les chaînes de pipeline et les arêtes *E* représentent les dépendances entre elles. Avec le parallélisme inter-opérateur, certaines chaînes de

pipeline sont exécutées simultanément sur des nœuds différents. Ce parallélisme est représenté par chaque chemin racine-feuille de l'arbre de la chaîne de pipeline (Lanzelotte et al., 1994).

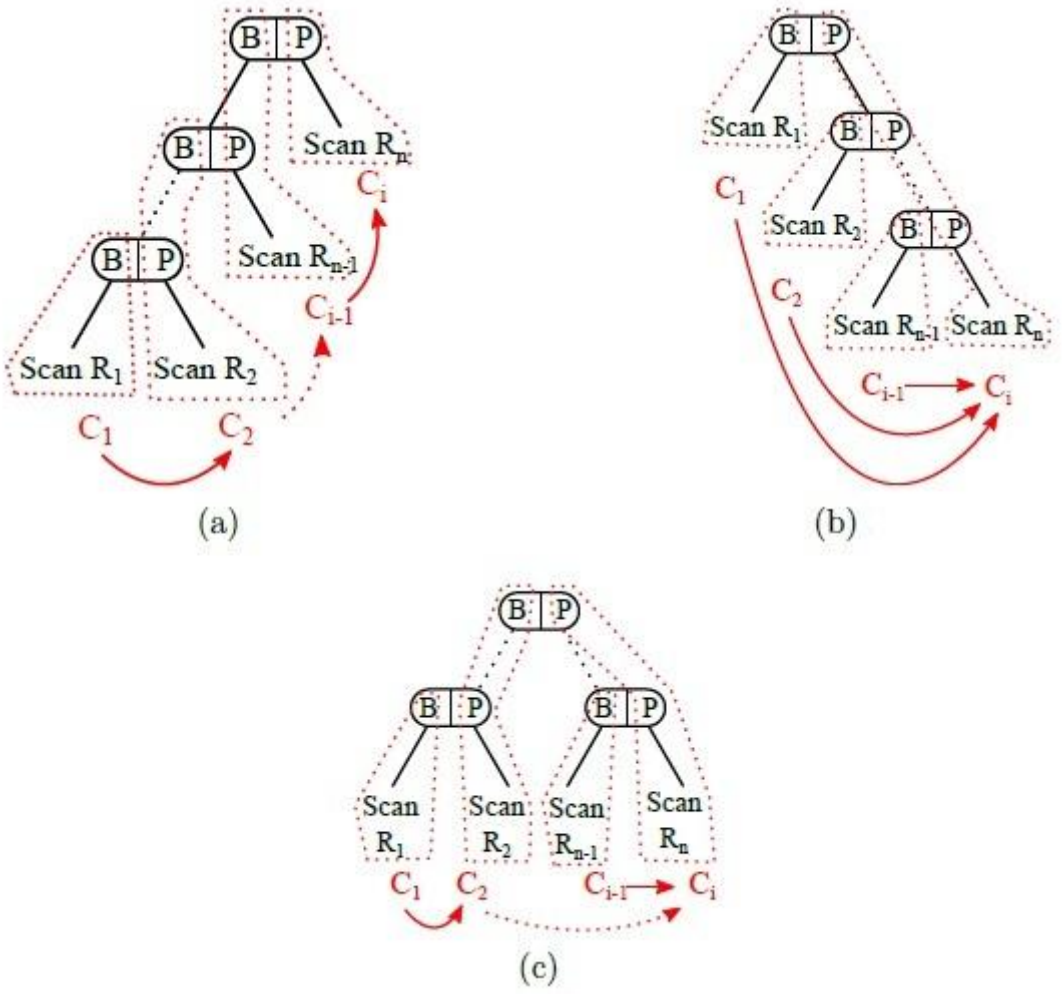

FIGURE 4.7 - Identification des chaines de pipeline dans les plans d'exécution
(a) linéaire gauche, (b) linéaire droit et (c) ramifié.

L'estimation du temps de réponse d'une requête peut être formulée comme suit : Soit C = {C_1, C_2,..., C_i} l'ensemble des chaînes de pipeline dans *PEQ*. Chaque chaine de pipeline C_i contient un certain nombre d'opérateurs exécutés en pipeline, c'est à dire C_i = {$O_{i,1}$, $O_{i,2}$,…, $O_{i,n}$}. Soit W = {W_1, W_2, ..., W_m} l'ensemble des chemins racine-vers-feuille pour tous les l nœuds feuille d'un arbre Y <V, E> dans un *PEQ* donné. Une branche d'exécution dans un plan de requête peut être constituée d'un certain nombre de chaînes de pipeline exécutées successivement, comme illustré à la Figure 4.8 (Tos, 2017).

Le temps de réponse estimé RT_Q de la requête Q est donné dans la Formule (4.3) (Tos, 2017). Il correspond au temps de réponse estimé de la branche d'exécution la plus longue $T(V_l)$ dans l'arbre de la chaîne de pipeline Y <V, E>.

$$RT_Q = MAX_m \sum_1^l T(V_l) \mid V_l \in W_m \qquad (4.3)$$

Soit $O_{i,p}$ est le $p^{ème}$ opérateur dans la chaine de pipeline C_i composée d'un certain nombre d'opérateurs $\{O_{i,1}, O_{i,2},\ldots, O_{i,p} \ldots, O_{i,k}\}$, exécutés en pipeline. Ces opérateurs ne commencent pas nécessairement le traitement en même temps. Par exemple, un opérateur $O_{i,2}$ peut dépendre du premier tuple du résultat généré par l'opérateur $O_{i,1}$. Par conséquent, pour les opérateurs autres que le premier, il existe un délai de traitement en pipeline égal à la durée d'attente mentionnée *pipeDelay* $O_{i,k}$ dans la Formule (4.4) (Lanzelotte et al., 1994).

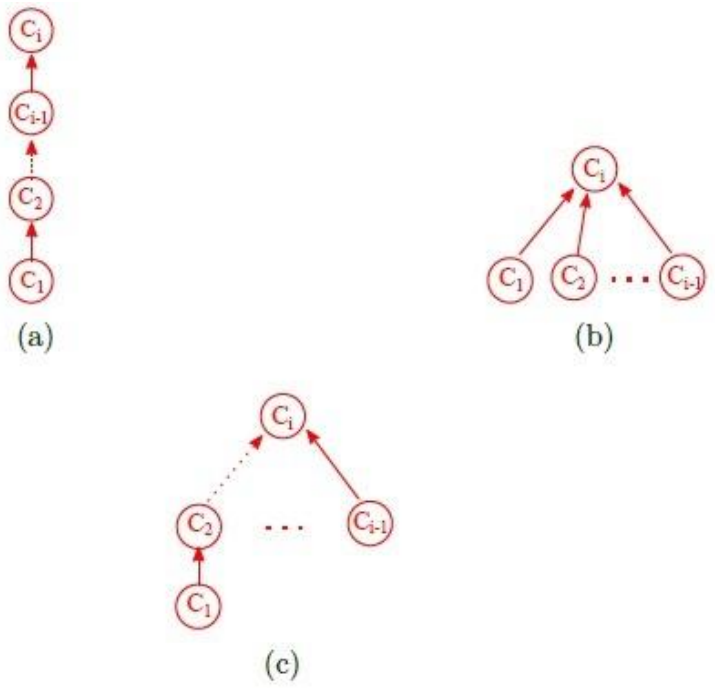

FIGURE 4.8 - Dépendances entre les chaines de pipeline dans un plan d'exécution (a) linéaire gauche, (b) linéaire droit et (c) ramifié.

Afin d'estimer le temps de réponse de chaque chaîne de pipeline RT_{Ci}, il faut alors estimer ce délai de pipeline de chaque opérateur $O_{i,p}$ dans la chaîne de pipeline C_i (Lanzelotte et al., 1994). L'exécution de C_i commence par le traitement de $O_{i,1}$. En conséquence, le temps de démarrage de $O_{i,2}$ dépend du temps de réponse de $O_{i,1}$. Le délai de pipeline d'un opérateur $O_{i,k}$ dépend de l'opérateur précédent $O_{i,k-1}$, et ainsi de suite.

$$RT_{Ci} = MAX_{\,k}\,(RT_{Oi,k} + pipeDelayO_{i,k}) \tag{4.4}$$

L'estimation du temps de réponse de C_i inclut alors de manière récursive le délai de traitement en pipeline de chaque opérateur en revenant du dernier opérateur au premier puisque le délai de traitement en pipeline du premier opérateur dans une chaîne de pipeline est égal à zéro, comme le montre la Figure 4.9. Nous pouvons alors déterminer de manière récursive les délais de pipeline jusqu'à ce que nous atteignions $O_{i,1}$.

$$RT_{Oi,p} = MAX\,[(CPU\_\ O_{i,p}\ \_Cost + IO\_\ O_{i,p}\ \_Cost),\ Transf\_\ O_{i,p}\ \_Cost] \tag{4.5}$$

L'estimation du temps de réponse de l'opérateur $O_{i,p}$ est donnée dans la Formule (4.5). Elle se base sur l'estimation des resources de calcul ($CPU\_\ O_{i,p}\ \_Cost$) , d'E/S ($IO\_\ O_{i,p}\_Cost$) et de transfer de données ($Transf\_\ O_{i,p}\_Cost$) pour chaque opérateur $O_{i,p}$ (Tos, 2017). Cette

estimation suppose que l'unité centrale peut commencer à traiter une relation transférée d'un nœud distant vers un nœud local dès qu'une quantité significative de données, par exemple une page, devient disponible localement. L'exécution d'un opérateur peut alors commencer avant le transfert de toutes les données, ce qui justifie l'utilisation de l'opérateur *Maximum* (*MAX*) dans la Formule (4.5).

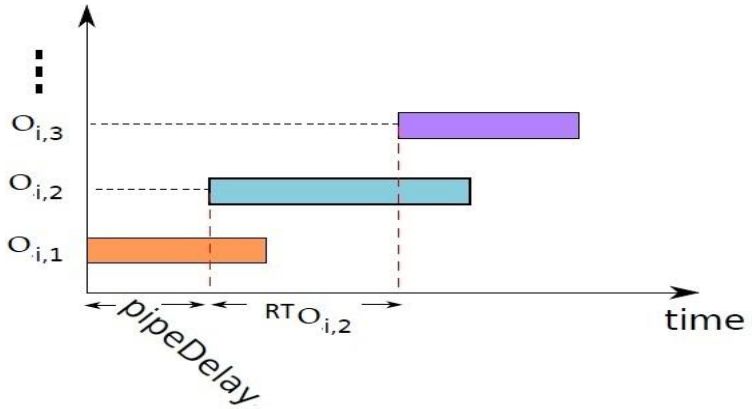

FIGURE 4.9 - Délai de traitement en pipeline.

4.4.3 Exemple d'estimation du temps de réponse d'un plan d'exécution linéaire droit

Dans ce qui suit, nous estimons le temps de réponse d'une requête relationnelle *Q* composée de plusieurs opérateurs de jointure, c'est-à-dire que *Q* est une requête multi-jointures. Pour cela, nous estimons les coûts *CPU_Q_Cost*, *IO_Q_Cost* et *Transf_Q_Cost*. Ensuite, nous mettons en évidence certains facteurs importants qui ont un impact sur cette estimation, notamment l'exécution de requêtes concurrentes.

Soit *PEQ* : <*Q*, *NEQ*> avec *Q* : <O_0, O_1, ..., O_{n-1}> un ensemble de *n* opérateurs de jointures et *NEQ*, un ensemble de nœuds répartis sur des CDs situés dans une même région RG_i. *NEQ* contient les nœuds qui exécutent les opérateurs de *Q* ainsi que les nœuds contenant les relations requises par ces opérateurs et leurs répliques existantes. *Q* est le résultat de la jointure de (*n* + *1*) relations (R_0, R_1, ..., R_n), comme illustré dans la Figure 4.10. Dans ce qui suit, nous nous concentrons sur l'estimation du temps de réponse d'une requête dont le plan d'exécution est un arbre linéaire droit, comme le montre la Figure 4.10. En d'autres termes, *Q*= (R_0 *Join* (R_1 *Join* (R_2 *Join* (… *Join* (R_{n-1} *Join* R_n)…). Ce type de plan d'exécution offre le meilleur potentiel pour l'exploitation du parallélisme. La taille des relations issues du *Build* (B_0, B_1,…, B_{n-1}) peut être prédite avec plus de précision puisque les estimations de cardinalité sont basées sur des prédicats appliqués aux relations de base.

Pour chaque opérateur de jointure, nous nous basons sur l'algorithme de jointure par hachage simple. Nous supposons que chaque nœud possède une mémoire suffisante pour contenir les tables de hachage construites pour chaque opérateur de jointure. Nous supposons également que l'impact du débordement de mémoire (Bucket Overflow) n'affecte pas les performances du processeur. Aussi, nous supposons que les relations intermédiaires ne sont pas matérialisées durant l'exécution d'un opérateur. Lors de l'estimation du temps de réponse de *Q*, le modèle de coûts s'appuie sur certaines informations telles que des

statistiques sur les relations, le nombre de tuples et leurs tailles, disponibles dans le catalogue (métabase) d'un SGBD. Dans ce contexte, nous nous basons sur certains travaux, par exemple (Tomov et al., 2004), pour obtenir une estimation précise de la sélectivité ainsi que la taille des relations intermédiaires.

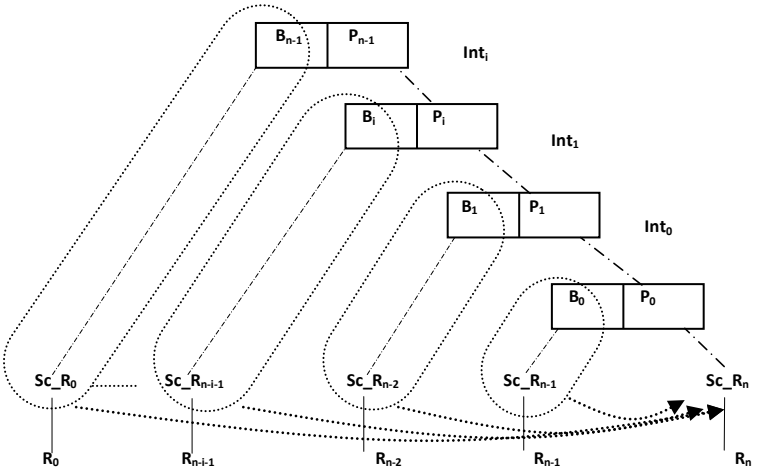

FIGURE 4.10 - Exemple de graphe de dépendances dans un plan d'exécution linéaire droit d'une requête relationnelle.

4.4.3.1 Estimation du temps de calcul

Soient les notations suivantes : NT_i le nombre de tuples de R_i, Si la taille d'un tuple de Ri (en octets). *Iread*/*Iwrite* le nombre d'instructions nécessaires pour lire/écrire une page de données depuis/sur un disque, *PSize* le nombre de tuples par page, I_{Tuple} le coût d'extraction d'un tuple d'une page en mémoire, $I_{Build/Probe}$ le nombre d'instructions de CPU dont chaque tuple a besoin lors de la construction/sondage de la table de hachage, NT_{Bi} la taille de la relation produisant le *Build* de la $i^{ème}$ jointure et NT_{Pi} la taille de la relation *Probe* issue de cette jointure. Enfin, T_{CPU} correspond à la durée moyenne d'une instruction CPU dans *NEQ*.

Toutes les lectures (*Scan*) des différentes relations (Sc_R_i) suivies par les opérateurs de construction de la table de hachage B_i (entourées par des lignes pointillées sur la Figure 4.10) sont indépendantes. Les tables de hachage peuvent alors être construites en parallèle afin de produire les différents B_i. Il s'agit du parallélisme indépendant. En conséquence, seul le chemin le plus long (Sc_R_{n-i-1} suivi de B_i) est pris en compte lors de l'estimation du coût en termes de CPU correspondant à la construction des différentes tables de hachage. Cela justifie l'utilisation de l'opérateur *Maximum* (*MAX*) dans la première partie de la Formule (4.6) (Hsaio et al., 1994). L'opérateur Probe *Pi* commence après la fin de P_{i-1}. En conséquence, NT_{Pi} est la taille de la (i-1) ème opération de jointure, à l'exception de $NT_{P0} = NT_n$ sur la Figure 4.10. Rappelons que chaque P_i consomme la sortie du Bi correspondant. Le temps de calcul estimé lors de l'exécution de n jointures est alors équivalent au temps de réponse estimé de la branche d'exécution la plus longue tels qu'illustré dans la Figure 4.8 (b). Le temps de calcul est alors estimé comme suit :

$$CPU\_Q\_Cost = [MAX_{i=0}^{n}\left(\left({}^{NT_i}/_{P_{size}} \times I_{read}\right) + \left(NT_i \times I_{Tuple}\right) + \left(NT_{Bi-1} \times I_{Build}\right)\right) + \sum_{i=0}^{n-1}\left(NT_{Pi} \times I_{Probe}\right)\Big] \times T_{CPU} \times (1 + \alpha + L) \quad (4.6)$$

avec $\alpha > 0$, un facteur de pondération qui tient compte des capacités matérielles des différents nœuds N_{ijk} (homogènes dans un CD), par exemple le taux de traitement, le débit moyen d'E/S dans le disque et les capacités de mise en cache. Soit L le facteur moyen de charge qui prend en compte la charge de travail de chaque N_{ijk} concerné par l'exécution de Q. Nous discutons la charge de travail $Load_{ijk}$ dans un nœud N_{ijk} dans la section 4.4.4.

L'estimation du temps de réponse pour un arbre linéaire gauche ou ramifié (Bushy) ne nécessite que quelques ajustements. Par exemple, dans un arbre linéaire gauche, NT_{Bi} est égal à la taille du dernier opérateur de jointure (NT_{Pi-1}), à l'exception de la génération du premier *Build* (égal à NT_0).

4.4.3.2 Estimation du temps de lecture des relations

La consommation des ressources d'E/S est également prise en compte lors de l'estimation du temps de réponse de la requête Q.

Lors de l'exécution de Q, les (n+1) relations requises sont lues à partir des disques des nœuds hébergeant ces relations. Que ce soit pour transmettre une relation à un autre nœud ou pour traiter la relation localement, le même modèle de consommation d'E/S est utilisé pour estimer la composante de temps de réponse liée aux opérations d'E/S sur chaque nœud. Le temps nécessaire à la lecture dépend de la taille des données lues, de la mémoire de stockage locale et du débit d'E/S spécifique à ce nœud, c'est-à-dire combien d'octets peuvent être lus par unité de temps.

Soit t_{pio} le temps de service du disque par page. Le coût estimé d'E/S de Q est donné dans la Formule (4.7).

$$IO\_Q\_Cost = \sum_{i=0}^{n} \left(\frac{NT_i \, X \, S_i}{P_{Size}}\right) \times t_{pio} \tag{4.7}$$

Rappelons qu'une requête peut nécessiter la lecture d'une relation particulière plusieurs fois au cours d'une exécution. Un calcul réaliste des coûts de stockage ne doit pas seulement tenir compte de la taille des relations, mais également du nombre d'opérations de lecture sur disque causées par l'accès à cette relation. En conséquence, cette simplification a été faite dans le but d'estimer rapidement *IO_Q_Cost*.

4.4.3.3 Estimation du temps de transfert de données

Le temps de transfert de données constitue une part importante du temps de réponse global d'une requête répartie. Lorsqu'un nœud participe au traitement d'une telle requête, on souhaite idéalement que ce nœud dispose localement des données pertinentes. Néanmoins, certaines données requises par les différents opérateurs de Q peuvent être présentes sur des nœuds distants. Aussi, certaines relations peuvent être répliquées lors de la satisfaction du SLA (scénarios 1 et 2 lors du placement de répliques). Cela engendre la consommation de ressources en termes de bande passante du réseau. Evidemment, le temps nécessaire au transfert de ces données est proportionnel au volume des données transférées.

$$Transf\_Q\_Cost = [\sum_{i=0}^{n-1} (NT_{Pi} \times S_{Inti}) + (NT_n \times S_n)]/NB \tag{4.8}$$

La bande passante disponible entre deux nœuds a un impact direct sur le temps de transfert de données entre ces deux nœuds. Soit *NB* la bande passante moyenne (octets/s) disponible entre les nœuds ∈ *NEQ* exécutant les différents opérateurs de jointure. Par ailleurs, chaque relation intermédiaire Int_i résultant d'un Probe *Pi* est toujours envoyée au nœud qui exécute la prochaine jointure (excepté le dernier P_{n-1} qui produit le résultat final). Soit S_{Int} la taille (en octets) d'un tuple d'une relation *Inti*. L'estimation du coût de transfert de données en termes de temps est indiquée dans la Formule (4.8).

4.4.4 Prise en compte des requêtes concurrentes

Dans la section précédente, nous avons vu que certains facteurs tels que le volume de données ou la bande passante entre nœuds (appartenant à *NEQ*) constituent des facteurs importants qui ont un impact significatif sur l'estimation du temps de réponse d'une requête *Q*. Dans ce qui suit, nous considérons également la charge de travail de chaque nœud exécutant un certain nombre d'opérateurs.

Soit $Load_{ijk}$ la charge de travail estimée sur un nœud N_{ijk}. Il prend en compte le taux d'arrivée AR sur N_{ijk} (AR_{ijk}) qui correspond au nombre d'opérateurs exécutés simultanément dans N_{ijk}. Ainsi, $Load_{ijk}$ dépend de la somme des coûts de calcul de tous les opérateurs O_q exécutés ou en attente d'exécution, c'est-à-dire dans la mémoire tampon, sur N_{ijk} comme indiqué dans la Formule (4.9).

$$Load_{ijk} = \sum_{q=1}^{\boldsymbol{AR_{ijk}}} (CPU\_O_q\_Cost) \times C \tag{4.9}$$

avec $0 < C < 1$, le facteur de complexité dans N_{ijk}. Il dépend de différents paramètres tels que le nombre de jointures exécutées dans N_{ijk}, qui résulte de la construction de plusieurs tables de hachage dans N_{ijk}. *CPU_Oq_Cost* correspond à l'estimation du coût de l'exécution de l'opérateur O_q. Son estimation (en secondes) dépend essentiellement du nombre d'instructions *#Inst* requis pour traiter O_q, comme indiqué dans la Formule (4.10).

$$CPU\_O_q\_Cost_{ijk} = \#Inst \times T_{CPU} \times (1 + \alpha) \tag{4.10}$$

4.5 MODELE ECONOMIQUE POUR L'EVALUATION DE REQUETES RELATIONNELLES

Les fournisseurs louent des services aux locataires en contrepartie d'une compensation monétaire (Foster et al., 2008). Ainsi, un fournisseur vise à générer des bénéfices lors de l'exécution des requêtes des locataires, tout en respectant les objectifs SLO pour ces locataires. L'impact économique de chaque décision doit alors être pris en compte.

Dans notre proposition, la création d'une nouvelle réplique intervient uniquement dans le but de satisfaire les objectifs du locataire, par exemple SLO_{RT}, afin d'éviter le paiement de pénalités par le fournisseur à son locataire. Néanmoins, une nouvelle réplique est créée seulement et seulement si elle est profitable pour le fournisseur. Dans cette perspective, le modèle de coûts proposé précédemment doit être étendu afin d'intégrer les coûts économiques liés à l'exécution d'une requête d'un locataire. Le modèle de coûts résultant dit 'modèle de coûts économique' permet d'estimer le profit économique du fournisseur *Q_Profit* lorsqu'une réplication est envisagée. Cela passe par l'estimation des revenus

monétaires (*Q_Revenues*) et des dépenses monétaires (*Q_Expenses*) du fournisseur pour l'exécution de *Q* dans un contexte multi-locataires.

On parle alors d'une réplication profitable si et seulement si les revenus monétaires estimés du fournisseur (*Q_Revenues*) sont supérieurs à ses dépenses monétaires estimées (*Q_Expenses*) lorsqu'une réplication est envisagée.

4.5.1 Environnement multi-locataires

De toute évidence, un fournisseur de Cloud met souvent en place un partage des ressources entre plusieurs locataires afin d'augmenter son profit économique. Cela se fait en consolidant diverses applications utilisées par plusieurs locataires sur un même serveur, c'est-à-dire que différents locataires exécutent leurs tâches simultanément sur un même serveur physique (Long et al., 2014). Par exemple, des milliers de locataires peuvent simultanément soumettre leurs requêtes à un SGBD. Les ressources disponibles sont alors mutualisées entre ces locataires (Tan and Babu, 2016; Sousa et al., 2018).

Dans un environnement multi-locataires (*multi-tenancy*), les performances d'un système dépendent non seulement de la charge de travail, des capacités matérielles mais également du nombre de locataires servis par un fournisseur. Les ressources dont dispose le fournisseur doivent alors être allouées de manière élastique en fonction de la demande de chaque locataire. En retour, chaque locataire paie le loyer des ressources qu'il utilise au fournisseur selon le modèle 'pay as you go' (Armbrust et al., 2010). Cela signifie qu'un locataire ne paie que ce qu'il consomme comme ressources. Rappelons qu'en plus de la période de facturation pendant laquelle le locataire loue des ressources auprès du fournisseur, le contrat SLA inclut d'autres paramètres qui peuvent être spécifiques à chaque locataire. Certains paramètres sont préalablement négociés entre le fournisseur et ses locataires. On citera notamment le montant de ce loyer payé par le locataire au fournisseur, le montant des pénalités payées par le fournisseur au locataire en cas de violation du SLA ou encore les exigences du locataire en termes de d'objectifs SLO.

4.5.2 Estimation du profit du fournisseur

Dans cette sous section, nous estimons les revenus et les dépenses du fournisseur lors de l'exécution d'une requête afin de déduire le profit économique du fournisseur. Nous rappelons que ces estimations sont faites uniquement lorsqu'une réplication est envisagée.

4.5.2.1 Estimation des revenus

Nous nous sommes intéressés à deux types de revenus perçus par le fournisseur : (i) les revenus générés par la location des ressources à ses locataires lors du traitement de leurs requêtes, et (ii) les revenus générés par la location d'un service particulier à un autre fournisseur. En effet, il est possible qu'un fournisseur de Cloud procède à la location de services à d'autres fournisseurs de Cloud. Néanmoins, nous supposons que la majeure source de revenus du fournisseur provient des locataires. Ces revenus sont décrits ci-dessous :

(i) Un montant de service (T_i*_Amount*) reçu de la part de chaque locataire T_i servi par ce fournisseur, pour les ressources qui lui ont été allouées. Certains fournisseurs

commerciaux, par exemple Google Cloud[21], facturent une location mensuelle des ressources (pour les abonnés) tandis que d'autres fournisseurs facturent pour une période plus courte, par exemple une heure pour l'utilisation de ressources de calcul avec OVH Cloud[22]. D'autres fournisseurs facturent l'exécution d'un certain nombre de requêtes, convenu dans le SLA. Par exemple, BigQuery[23] de Google applique un tarif pour un certain nombre de requêtes, en fonctions du volume de données traitées. Dans notre cas, nous avons également considéré une tarification pour un nombre maximum de requêtes # Q pendant la période de facturation, par exemple 0,6 $ pour l'exécution de 1 000 requêtes. Ceci correspond à une tarification fixe. D'un autre coté, le prix peut être déterminé en fonction de l'offre et la demande, ce qui correspond à une tarification dynamique. Cela permet au fournisseur de mieux exploiter le potentiel de paiement des locataires et donc de réaliser un plus grand bénéfice.

(ii) Le prix probable d'un crédit de service (*Rent_XaaS*) reçu lorsque le fournisseur loue un service XaaS donné à un autre fournisseur (Serrano et al., 2016).

Nous estimons alors les revenus du fournisseur par requête (*Q_Revenues*) comme indiqué dans la Formule (4.11). Il est alors évident que le fournisseur doit définir le montant de service T_i*_Amount* en fonction de toutes ses dépenses afin d'obtenir un gain raisonnable. Rappelons qu'un fournisseur peut créer une ou plusieurs répliques afin de satisfaire l'objectif d'un locataire, ici SLO_{RT}. Néanmoins, un locataire n'est pas facturé pour le nombre de répliques créées lors de l'exécution de sa requête.

$$Q\_Revenues = (\sum_1^n (Ti\_Amount)/\#Q) + Rent\_XaaS \quad (4.11)$$

Il convient aussi de noter qu'il n'est pas réaliste de connaitre à l'avance le nombre de requêtes soumises par un locataire et avec quel taux d'arrivée. *Q_Revenues* correspond alors à l'estimation du revenu moyen attendu pour l'exécution d'une requête suivant les modalités fixées dans le SLA, par exemple en ayant le nombre maximum de requêtes traitées durant une période de facturation. L'estimation décrite ci-dessus permet alors de prédire rapidement le revenu moyen par requête avec une précision suffisante pour la décision de réplication.

4.5.2.2 Estimation des dépenses

Par rapport à l'estimation des revenus du fournisseur lors de l'exécution d'une requête *Q* d'un locataire, l'estimation des dépenses est plus complexe. Ces dépenses correspondent à la somme des prix des ressources allouées lors de l'exécution de *Q*. En effet, le fournisseur doit payer périodiquement le coût de fonctionnement de chaque serveur qui met à disposition ces ressources, par exemple en termes de stockage et de bande passante.

La création d'une nouvelle réplique engendre des coûts de transfert et de stockage supplémentaires afin de satisfaire le contrat SLA, ce qui évite au fournisseur le payement de pénalités au locataire. Rappelons qu'un locataire ne doit pas être facturé pour la création d'une réplique. Le coût engendré par toute nouvelle création de réplique doit alors être inclus dans les dépenses du fournisseur afin de pouvoir décider de la profitabilité de la

[21] https://cloud.google.com/products/compute?hl=fr
[22] https://www.ovhcloud.com/fr/public-cloud/prices/#compute
[23] https://cloud.google.com/bigquery/pricing?hl=fr

réplication de données. Evidemment, les dépenses du fournisseur augmentent lorsque le nombre de répliques augmente. Une stratégie de réplication efficace doit alors envisager un ajustement dynamique du facteur de réplication comme décrit précédemment. Cela permet de réduire les dépenses du fournisseur.

Dans un environnement Cloud, l'exécution d'une requête Q peut impliquer la participation d'un certain nombre de nœuds. Les ressources disponibles sur ces nœuds peuvent être entièrement ou partiellement utilisées lors de cette exécution. Elles coûtent inévitablement de l'argent au fournisseur (Greenberg et al., 2009) qui les facturent à son tour aux locataires qu'il sert. Ces ressources incluent notamment le coût de traitement par les processeurs (C_{CPU}), le coût du stockage (C_S) et le coût de la bande passante du réseau (C_{NB}). Ces dépenses doivent également inclure le coût probable des pénalités (C_P) payées aux locataires et les coûts d'estimations du temps de réponse et du profit du fournisseur (C_{Est}) engendrés par la réplication. Nous incluons également dans les dépenses du fournisseur : les coûts de la consommation énergétique (C_{Energ}), les coûts d'investissement du fournisseur (C_{Inv}) ainsi que les coûts de location de services *XaaS* auprès d'autres fournisseurs (C_{XaaS}). L'estimation du coût monétaire total pour l'exécution d'une requête par le fournisseur revient alors à l'estimation de l'ensemble de dépenses monétaires citées dans la Formule (4.12).

$$Q\_Expenses = C_{CPU} + C_S + C_{NB} + C_P + C_{Est} + C_{Energ} + C_{Inv} + C_{XaaS} \quad (4.12)$$

L'estimation de chaque dépense lors de l'exécution d'une requête Q est décrite en détail ci-dessous. Nous utilisons les paramètres suivants : Soit $\#MV$ le nombre de nœuds exécutant les opérateurs de Q. Soit S_{dl} la taille d'une relation stockée sur un nœud local. S_{dr} correspond à la taille d'une relation distante transférée entre deux nœuds. Cette relation peut être une réplique nouvellement créée. d'/r' correspond au nombre de relations locales/distantes requises pour l'exécution de Q. Il convient de signaler que d' et r' incluent également le nombre de répliques requises pour l'exécution de Q.

$$Q\_Expenses = \sum_{q=1}^{\#MV}(RT_{Oq} \times CPU\_Cost) + \sum_{l=1}^{d'}(S_{dl} \times Stor\_Cost_{ij}) + \sum_{r=1}^{r'}(S_{dr} \times Netw\_Cost) + Avg\_Pen\_Cost + Estim\_Cost + Energ\_Cost + Inv\_Cost + XaaS\_Cost \quad (4.13)$$

Coût des traitements (C_{CPU}). Les opérateurs d'une requête peuvent être traités conjointement par plusieurs nœuds dans un environnement distribué. Par conséquent, ce coût monétaire correspond au coût de l'ensemble des CPU sur les $\#MV$ participants à l'exécution de Q comme indiqué dans la Formule (4.13). Soit RT_{Op} le temps de réponse estimé pour l'exécution d'un opérateur $O_q \subseteq Q$. Soit CPU_Cost le coût, en millions d'instructions par unité de temps, par exemple une heure dans le Cloud d'Amazon, pour utiliser un nœud alloué pour l'exécution de O_q.

Coût du stockage (C_S). Le coût monétaire de stockage dans la Formule (4.13) inclut non seulement le coût de stockage des relations de base mais également le coût de stockage des répliques de chaque relation. D'un autre coté, nous prenons en compte l'hétérogénéité de tarification des ressources entre les CDs. Par exemple, chaque CD_{ij} a son propre tarif de stockage. Dans la Formule (4.13), le stockage d'une relation/réplique a un coût monétaire $Stor\_Cost_{ij}$ en fonction du prix appliqué dans CD_{ij} qui l'héberge. Enfin, nous avons supposé

que les relations intermédiaires des opérations en pipeline ne sont pas stockées. Ceci explique pourquoi leur coût de stockage n'est pas inclut dans la Formule (4.13).

Coûts de transferts des données (C_{NB}). Le traitement des requêtes dans des environnements distribués entraîne inévitablement la consommation de la bande passante du réseau en raison du transfert de données entre nœuds.

Le coût unitaire de transfert de données dans un CD est moins coûteux que le transfert de données entre CDs. En conséquence, le coût des transferts de données dépend de l'emplacement des nœuds (transfert intra-CD ou entre CDs $\in RG_i$ dans notre cas). Dans la Formule (4.13), *Netw_Cost* représente le coût monétaire moyen des transferts de données lors de l'exécution de *Q*. Évidemment, il inclut le coût de transfert de données lors de la création d'une nouvelle réplique.

Coûts des pénalités (C_P). Les pénalités probables payées par le fournisseur à ses locataires doivent également être prises en compte lors de l'estimation des dépenses du fournisseur. Dans la Formule (4.13), *Avg_Pen_Cost* correspond aux pénalités probables payées au locataire T_i lorsque l'objectif SLO_{RT} n'est pas satisfait.

Bien que le fournisseur prend les précautions nécessaires afin d'éviter le paiement d'une pénalité, par exemple à travers la réplication de données, il se peut que certaines requêtes ne répondent pas à l'objectif SLO_{RT}. Dans ce cas, il se doit de payer une pénalité à T_i. Dans notre proposition, nous nous appuyons sur le coût monétaire moyen de pénalité par requête, payé par le fournisseur à ses locataires lors de la précédente période de facturation.

Coûts des estimations (C_{Est}). Comme décrit dans les sections précédentes, la prise de décision de réplication s'appuie sur l'estimation du temps de réponse d'une requête ainsi que l'estimation des revenus et dépenses du fournisseur pour l'exécution d'une requête lorsque la création d'une réplique est envisagée. Ces estimations ont évidemment un coût que nous intégrons dans l'estimation des dépenses du fournisseur. Ce coût est noté par *Estim_Cost* dans la Formule (4.13).

Coûts de la consommation énergétique (C_{Energ}). Comme énoncé dans le chapitre 3, la consommation en électricité représente un coût important pour les fournisseurs de Cloud. Ces derniers doivent alors intégrer le coût de la consommation énergétique *Energ_Cost* dans leurs dépenses comme le montre la Formule (4.13). La consommation énergétique engendrée par une réplication de données doit également être prise en compte. Dans les travaux que nous avons décrits dans ce manuscrit, ce coût est fixé d'avance de manière statique. Actuellement, nous co-encadrons une Thèse à l'IRIT (Séguéla et al., 2019a) qui vise à modéliser cette consommation énergétique tout en l'intégrant à la fonction objectif visée par une stratégie de réplication de données.

Coûts des investissements (C_{Inv}). En réalité, les fournisseurs de Cloud payent de nombreux types de dépenses, allant des salaires des employés à la sécurité physique du matériel. Ces coûts ne sont pas directement liés à la réplication de données ou à l'exécution des requêtes. Cependant, le fournisseur voudrait naturellement toujours avoir une marge bénéficiaire suffisante pour couvrir ces coûts. Ainsi, deux manières sont possible pour inclure le coût des investissements dans les dépenses du fournisseur lors de l'exécution d'une requête d'un locataire : (i) le profit monétaire du fournisseur doit être supérieur à un seuil de profit *Profit_Th*, c'est-à-dire *Profit_Q* > *Profit_Th* tels que *Profit_Th* correspond à un seuil de marge bénéficiaire prédéfini. Ce seuil peut être vu comme étant la montant minimal de profit qu'un

fournisseur souhaite générer lors de l'exécution d'une requête, ou (ii) le profit monétaire du fournisseur doit être positif lors de l'exécution d'une requête, c'est-à-dire $Profit\_Q > 0$, tout en incluant le coût des investissements C_{Inv} dans les dépenses du fournisseur.

Dans notre cas, nous avons opté pour la deuxième solution en incluant le prix de l'investissement du matériel et des licences logicielles dans *Inv_Cost* comme le montre la Formule (4.13). Enfin, lorsque le fournisseur loue un service XaaS donné auprès d'un autre fournisseur, ses dépenses en investissement pourraient également inclure le prix de la location de ces services (*XaaS_Cost*).

4.5.3 Gestion des pénalités

Lorsque le fournisseur exécute une requête *Q* avec un temps de réponse effectif supérieur au seuil de temps de réponse *RT_SLO_PSQ*, une pénalité monétaire est versée par le fournisseur à son locataire. Nous nous concentrons uniquement sur la violation du SLA en termes d'objectif de temps de réponse. Ainsi, un payement de pénalité intervient uniquement lorsque le temps de réponse effectif d'une requête dépasse le seuil de temps de réponse *RT_SLO_PSQ*, mentionné dans le SLA.

Soit RTE_Q le temps de réponse effectif de *Q*. Comme le montre l'algorithme de gestion des pénalités (Algorithme 4.3), lorsque RTE_Q est inférieur à *RT_SLO_PSQ* (c'est à dire $RTE_Q < RT\_SLO\_PSQ$), aucune pénalité n'est payée (ligne 2). Dans le cas contraire, nous procédons comme suit :

(i) Lorsque $RTE_Q \in [RT\_SLO\_PSQ, RT\_SLO\_PQ]$, le fournisseur accepte de payer un montant de pénalité *Pen_RT* à son locataire (ligne 3). La valeur de *Pen_RT* est définie dans le contrat SLA et correspond à la pénalité payée chaque fois que RTE_Q dépasse *RT_SLO_PSQ*.

(ii) Lorsque RTE_Q dépasse le seuil critique de temps de réponse, c'est-à-dire ($RTE_Q > RT\_SLO\_PQ$), le fournisseur paie également *Pen_RT* mais, en incrémentant la valeur de *V_Nber* qui correspond au nombre de fois que RTE_Q dépasse *RT_SLO_PQ*. Dans ce cas, on parle d'une violation critique du SLA.

Algorithme 4.3 - Gestion des pénalités.

Entrées : *T_i, RTE_Q, RT_SLO_PSQ, RT_SLO_PQ, Pen_RT, Data_Vol, Vol_SLO, Critical_V_Nber. Initialement, V_Nber=0.*
Sortie : *Payement éventuel de Pen_RT par le fournisseur à son locataire T_i.*

1. ***Begin***
2. ***if*** *$RTE_Q < RT\_SLO\_PSQ$* ***then*** *{Pas de pénalité à payer}*
3. ***else {if*** *($RTE_Q < RT\_SLO\_PQ$)* ***then*** *{Le fournisseur paye une pénalité Pen_RT à T_i}*
4. ***else {*** *Le fournisseur paye une pénalité Pen_RT à T_i; V_Nber++}*
5. ***if*** *((V_Nber = Critical_V_Nber) and (Data_Vol > Vol_SLO))* ***then*** *{Négociation d'un nouveau RT_SLO_PSQ avec T_i; V_Nber= 0}}*
6. ***End***

Des violations fréquentes du SLA nuisent à l'image du fournisseur et réduisent son profit économique. En conséquence, elles doivent être réduites autant que possible. Pour cela, nous proposons de limiter le nombre de violations critiques du SLA (*V_Nber*) en limitant le

nombre de ces violations (*V_Nber ≤ Critical_V_Nber).* Lorsque une violation critique est constatée (*Critical_V_Nber)* fois et que le volume total de données (*Data_Vol)* dépasse un seuil de volume de données convenu dans le SLA (*Vol_SLO)*, le fournisseur a le droit de renégocier la valeur du seuil de temps de réponse *RT_SLO_PSQ* (à la hausse) avec son locataire. En conséquence, il pourra également revoir à la hausse la valeur du seuil de temps de réponse critique *RT_SLO_PQ.* Cela n'intervient que lorsque les deux seuils *Critical_V_Nber* et *Vol_SLO* sont atteints simultanément (ligne 5 dans l'Algorithme 4.3). En effet, l'augmentation du volume des données engendre souvent une augmentation du montant de service dans les Cloud commerciaux. Réduire le nombre de violations de SLA réduit les coûts liés aux pénalités. En analysant les Formules (4.1) et (4.13), il est clair que réduire le coût des pénalités sans augmenter le coût de la réplication améliore le profit économique du fournisseur.

4.6 CONCLUSION

Dans ce chapitre, nous avons présenté nos propositions permettant une réplication dynamique de données qui satisfait les objectifs de disponibilité minimum SLO_{AV} et de temps de réponse SLO_{RT} pour les locataires, tout en prenant en compte le profit économique du fournisseur.

Maintenir un nombre minimum de répliques pour chaque relation permet de satisfaire l'objectif SLO_{AV}. Concernant la satisfaction de l'objectif des performances, le fournisseur propose à ses locataires un seuil de temps de réponse *RT_SLO_PSQ* comme garantie de performances au lieu d'une performance optimale. Grâce à un modèle de coûts que nous avons proposé, le temps de réponse du plan d'exécution parallèle de chaque requête *Q* d'un locataire est estimé avant l'exécution de celle-ci dans une région donnée. Une création d'une réplique est envisagée uniquement si l'objectif de temps de réponse SLO_{RT} n'est pas satisfait. Contrairement à la plupart des stratégies de réplication proposées dans la littérature, cette création intervient pour un ensemble de requêtes au lieu d'une réplication par requête, ce qui diminue significativement les coûts liés à la réplication.

Nous avons proposé une heuristique permettant de trouver un nœud de placement acceptable pour une nouvelle réplique, de telle sorte que l'objectif SLO_{RT} soit satisfait et que cette réplication soit profitable pour le fournisseur. Pour cela, nous avons étendu le modèle de coûts précédent en intégrant les coûts économiques liés à l'exécution d'une requête. Le modèle de coûts obtenu que nous avons appelé 'modèle de coûts économiques' permet l'estimation des revenus et des dépenses du fournisseur à chaque fois qu'une réplication est envisagée. Les coûts liés à la réplication ainsi que les pénalités éventuelles payées par le fournisseur à ses locataires ont été modélisés dans les dépenses du fournisseur. De plus, nous avons proposé un algorithme permettant d'éviter le payement répétitif de pénalités par le fournisseur à ses locataires, ce qui réduit les dépenses du fournisseur.

Enfin, le fournisseur doit éviter de créer des répliques inutiles. Nous avons alors proposé un algorithme qui ajuste dynamiquement le nombre de répliques afin de permettre une gestion élastique des ressources, ce qui permet au fournisseur de s'adapter à la variation de la charge de travail et donc, de s'adapter aux requêtes des locataires. Le prochain chapitre valide nos propositions.

CHAPITRE 5

Evaluation des performances

Résumé

Dans ce chapitre, nous évaluons les performances de nos propositions. D'abord, nous présentons l'environnement expérimental. Puis, nous comparons les performances de nos propositions à celles d'autres stratégies proposées dans les systèmes Cloud. Enfin, nous analysons et discutons les résultats obtenus.

Sommaire

5.1 INTRODUCTION

Les propositions présentées dans le chapitre précédent ont été validées à travers des évaluations expérimentales. Nous avons simulé un système Cloud exploitant des centres de données répartis à grande échelle. Nous avons opté pour une validation par simulation car celle-ci nous permet de contrôler directement certains paramètres afin d'analyser leur impact individuel sur les performances, par exemple le taux d'arrivée des requêtes et la variation de la configuration du système. La simulation permet également de mettre en place des scénarios de tests qui peuvent être répétés surtout lorsqu'il s'agit de comparer de nombreuses stratégies de réplication de données. Dans ce chapitre, nous avons choisi de se concentrer principalement sur les résultats obtenus avec les stratégies RSPC (Mokadem and Hameurlain, 2020) et PEPR (Tos et al., 2018). Dans ce qui suit, nous commençons par

présenter l'environnement expérimental. Puis, nous comparons les performances de nos solutions à celles de trois autres stratégies proposées dans les systèmes Cloud. Enfin, nous analysons les résultats obtenus.

5.2 ENVIRONNEMENT EXPERIMENTAL

Afin de valider nos propositions, nous avons utilisé CloudSim (Calheiros et al., 2010), un outil de simulation Open Source, permettant de simuler des MVs au sein de plusieurs CDs. Nous avons simulé un système Cloud composé de 3 régions, comme illustré dans la Figure 4.2. Au sein de chaque région, nous avons implémenté 10 CDs. Puis, 1000 MVs sont implémentées au sein de chaque CD.

Nous avons commencé par l'extension de CloudSim en implémentant la réplication de données, le placement de requêtes. Par ailleurs, des capacités de calcul, de stockage et de mémoire sont affectées à chaque MV, et des latences / bandes passantes hétérogènes sont simulées entre les CDs et les régions. Nous nous sommes basé sur (Barosso et al., 2018) pour établir les caractéristiques des différentes ressources afin de représenter un environnement Cloud de manière réaliste.

Paramètre	*Valeur*
Nombre de régions	3
Nombre de CDs par région	10
Nombre de MV par CD	1000
Taille moyenne d'une relation	700 Mb
Bande Passante (BP) moyenne disponible entre régions (délai respect.)	500 Mb/s (150ms respect.)
BP moyenne disponible entre CDs (délai respect.)	1Gb/s (50ms respect.)
BP disponible entre MVs $\in$ CD (délai respect.)	8 Gb/s (10 ms respect.)
Capacité moyenne de calcul par MV	1500 MIPS
Capacité moyenne de stockage par MV	10 Gb
Durée d'une Période de Facturation (PF)	10 min
#requêtes/ PF	[3000, 48000]
RT_SLO_PQ	180s
RT_SLO_PSQ	{50, 100, 150}s
Revenu du fournisseur/1000 requêtes (pour *RT_SLO_PSQ*)	{1, 0.8, 0.6}$
Coût moyen de stockage	0.15$/Tb
Coût de transfert Intra-CD	0.0005$/ Gb
Coût de transfert Inter-CD dans une même région	0.002$/Gb
Coût de transfert entre régions	0.07$/ Gb
Coût de traitement / million d'instructions (*CPU_Cost*)	$1\$/10^9$ MI
Montant de pénalité / violation (*Pen_RT*)	0.0025$
NSetQ	10
Repl_Fact_Min	4

TABLE 5.1 - Paramètres de configuration.

Différents concepts économiques ont également été pris en compte : (i) un prix monétaire est établi pour chaque ressource. Nous nous sommes basés sur la moyenne des prix appliqués par les fournisseurs commerciaux Google Cloud, AWS et Microsoft Azure[24], (ii) un locataire

[24] Amazon S3 Pricing. https://aws.amazon.com/s3/pricing/. Azure storage pricing. https://azure.microsoft.com/en-us/pricing/details/storage/. Azure data transfer pricing. https://azure.microsoft.com/en-us/pricing/details/data-transfers/. Google Cloud pricing https://cloud.google.com/compute/pricing?hl=fr. Juillet 2020.

est facturé pour un nombre donné de requêtes (ici, 1000 requêtes) et (iii) une violation du SLA est modélisée par un montant de pénalité payée par le fournisseur à ses locataires.

Nous avons utilisé un programme de génération de données[25] pour des requêtes du banc d'essai TPC-H[26]. Concernant la distribution des données, nous avons considéré : (i) une distribution uniforme et (ii) une distribution non uniforme, ici zipf (Bresslau et al., 1999), conçue pour réagir à la popularité des données et modéliser des accès sans contrainte d'une population indépendante telle que les utilisateurs d'Internet. Le facteur zipf (z) est défini à 1. Le taux d'arrivée des requêtes suit une distribution de Poisson.

Dans nos différentes expériences, un certain nombre de requêtes (3000, 12000, 30000 puis 48000) sont soumises au cours de chaque Période de Facturation (PF) de 10 minutes. Les cloudlets, associés aux requêtes, sont attribuées à des MVs choisies de manière aléatoire afin d'accéder aux relations elles-mêmes distribuées sur des MVs sélectionnées de manière aléatoire également. Nous avons choisi de traiter avec un sous-ensemble de requêtes fournies dans TPC-H {*Q4*, *Q10* et *Q8*} à des fins d'analyse. Ces requêtes ont des niveaux différents de complexité {1, 3 et 7 jointures respectivement}. Nous les appelons requêtes simples, intermédiaires et complexes respectivement.

Nous rappelons aussi que chaque requête soumise par un locataire est transformée en un plan d'exécution parallèle par l'optimiseur de requêtes. Pour l'ensemble des requêtes, nous avons retenu un plan d'exécution parallèle de type linéaire droit. Ce dernier est considéré comme une entrée pour nos simulations. Ensuite l'allocation de ressource est faite. Les coûts monétaires des ressources sont obtenus en se basant sur les différentes formules proposées dans le chapitre précédent. Les stratégies comparées sont entièrement codées en Java. Enfin, une relation de base de données en lecture seule constitue la granularité de la réplication. La Table 5.1 décrit les principaux paramètres utilisés dans notre configuration.

5.3 LES STRATEGIES COMPAREES

Nous comparons les performances des stratégies RSPC et PEPR à celles de trois autres stratégies de réplication proposées dans les systèmes Cloud. La stratégie RSPC a été largement décrite dans le chapitre 4. Les autres stratégies ont été décrites tout au long du chapitre 3. Ci-dessous, nous rappelons les principales caractéristiques de ces stratégies :

(i) Dans la stratégie CDRM (*the Cost-effective Dynamic Replication Management strategy*) (Wei et al., 2010), une réplique est placée sur le nœud ayant la plus faible probabilité de blocage. Cela permet de réduire la congestion due à l'accès aux données, ce qui améliore l'équilibrage de charge. Une probabilité de blocage est calculée sur chaque MV afin de mesurer sa charge de travail. Puis, les MVs surchargés sont bloquées pour la réception de nouvelles requêtes. Cette stratégie ne prend pas en compte les coûts économiques et la satisfaction du SLA n'est pas considérée comme un critère de décision pour la réplication de données.

(ii) La stratégie PEPR (*PErformance and Profit oriented data Replication strategy*) (Tos et al., 2018) tire profit, comme la stratégie RSPC, de la hiérarchie au niveau de la Bande Passante (BP) du réseau. Cela permet au fournisseur de réduire la consommation de ressources en termes de BP. PEPR prend en compte les

[25] https://www.microsoft.com/en-us/download/details.aspx?id=52430. Juillet 2020.

[26] *TPC-H benchmark specification*, 2019, [online]: http://www.tpc.org/tpch/

bénéfices du fournisseur lors de la création d'une nouvelle réplique. Cependant, la réplication est faite par requête. De plus, l'emplacement des répliques est basé sur la sélection du CD le moins cher qui satisfait SLO_{RT}.

(iii) La stratégie DCR2S (*Dynamic Cost-aware Re-replication and Re-balancing strategy*) (Gill and Singh, 2016) vise à créer une réplique pour les données dont la popularité dépasse un certain seuil. Les données sont re-répliquées à partir de CDs de coûts élevés vers des CDs moins chers afin de réduire le coût de la réplication, qui doit être absolument inférieur à un budget initial fixé à l'avance. Dans nos expériences, nous le définissons comme un pourcentage donné du coût d'exécution de la requête. Enfin, DCR2S néglige l'équilibrage de charge

(iv) La stratégie DPRS (*Dynamic Popularity aware Replication Strategy*) (Mansouri et al., 2017) ne réplique qu'une partie (20%) des données les plus populaires sur les meilleurs emplacements en fonction du nombre d'accès à ces données, de l'espace de stockage disponible et de la localisation d'un nœud sans pour autant considérer des aspects économiques tels que le profit du fournisseur. En dépit de l'exploitation de l'accès en parallèle aux données, il néglige l'effet du mode d'accès aux données lors de la réplication de données. De plus, la hiérarchie du point de vue de la BP n'est pas prise en compte.

5.4 RESULTATS OBTENUS

Nous avons évalué les performances des stratégies comparées en considérant différentes métriques. Dans ce qui suit, nous mesurons :

(i) Le facteur de réplication moyen par relation et le temps de réponse moyen d'une requête sous différentes distributions de données (uniforme et non uniforme) tout en variant le taux d'arrivée des requêtes.

(ii) L'ajustement du facteur de réplication notamment lors de la variation de la charge de travail.

(iii) L'impact de la configuration du système (nombre de CDs/régions) sur les performances.

(iv) L'impact de la complexité des requêtes sur les performances.

(v) Les dépenses du fournisseur incluant différentes ressources (en termes de stockage et de BP ainsi que les pénalités payées), ce qui nous permet de déduire le bénéfice du fournisseur.

Ces mesures sont prélevées/calculées à la fin d'une PF. Par ailleurs, nous avons expérimenté avec différentes valeurs de seuil de temps de réponse (non critique) *RT_SLO_PSQ*, allant d'une valeur stricte (50s) à une valeur souple (150s) en passant par une valeur intermédiaire (100s). Evidemment, les revenus du fournisseur dépendent de ces seuils, comme indiqué dans la Table 5.1. Nous appelons la stratégie RSPC sous ces seuils par *RSPC'50*, *RSPC'150* et *RSPC'100* respectivement. La valeur du seuil critique du temps de réponse *RT_SLO_PQ* est fixée à 180s. Ces valeurs sont établies sur la base d'expériences préliminaires.

5.4.1 Temps de réponse et facteur de réplication moyens

Les Figures 5.1 (a) et (b) montrent le facteur de réplication moyen par relation et le temps de réponse moyen par requête respectivement issus des stratégies de réplication lorsque la

distribution de données est uniforme. Le facteur de réplication dans RSPC n'inclut pas les répliques créées initialement pour satisfaire SLO_{AV} (ici, au nombre de 4). Aussi, nous utilisons des requêtes qui incluent de manière aléatoire des requêtes simples, intermédiaires et complexes.

Avec un nombre réduit de requêtes, par exemple moins de 12.000 requêtes/PF, les stratégies comparées génèrent un nombre réduit de répliques pour chaque relation (entre 4 et 9 répliques). Le facteur de réplication issu de CDRM, DCR2S et DPRS est plus important que le facteur issu de RSPC (dans ses trois options RSPC'50, RSPC'100 et RSPC'150). RSPC ne crée aucune nouvelle réplique lorsque SLO_{RT} est satisfait alors que CDRM vise à équilibrer la charge de travail entre les différents nœuds en créant davantage de répliques. DPRS fournit le meilleur temps de réponse tandis que RSPC'150 fournit le temps de réponse le plus élevé.

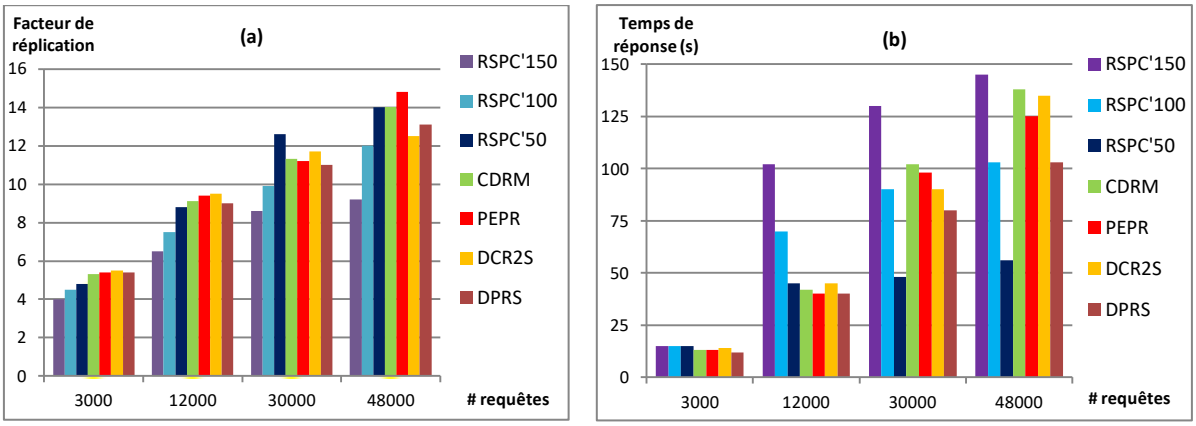

FIGURE 5.1 - Impact du taux d'arrivées de requêtes sur (a) le facteur de réplication moyen et sur (b) le temps de réponse moyen avec une distribution uniforme des données.

Lorsque le nombre d'accès aux données augmente, le temps de réponse augmente également. Avec 30.000 requêtes/PF (ce qui correspond à 50 requêtes/s), RSPC'50 présente le meilleur temps de réponse. RSPC'50 crée le plus important nombre de répliques afin de satisfaire SLO_{RT} alors qu'un nombre réduit de répliques est créé avec RSPC'150. Néanmoins, le temps de réponse issu de RSPC (les 3 options RSPC'150, RSPC'100 et RSPC'50) est inférieur à *RT_SLO_PSQ*. D'un autre coté, CDRM ne considère que la fréquence d'accès et ne crée pas plus de répliques lorsqu'un équilibrage de charge est atteint alors que DCR2S crée des répliques supplémentaires afin d'améliorer le temps de réponse tant que le budget limite n'est pas atteint.

Nous analysons également le comportement de chaque stratégie lors de pics de charge de travail. Un plus grand nombre de requêtes (48.000 requêtes/PF correspondant à 80 requêtes/s) engendre une augmentation du facteur de réplication, comme le montre la Figure 5.1 (a). RSPC'50, RSPC'100 et RSPC'150 ne nécessitent que quelques répliques supplémentaires. RSPC'150 ne dépasse pas le seuil RT_SLO_PSQ alors que les temps de réponse moyens issus de RSPC'100 et RSPC'50 ne dépassent que légèrement les seuils RT_SLO_PSQ respectifs. Certaines MVs sont bloquées pour la réception de nouvelles requêtes dans CDRM et les données populaires sont mises à jour avec DPRS, ce qui génère des répliques en dehors de la région recevant les requêtes. Par ailleurs, seules quelques répliques sont créées avec DCR2S. Une fois que le coût de la réplication dépasse le budget limite, l'algorithme de type 'sac à dos' tente d'optimiser le coût de la réplication en répliquant

sur des CDs à moindre coût. Cependant, l'équilibrage de charge n'est pas pris en compte et la réplication en dehors de la région locale n'améliore pas forcément le temps de réponse. Il en est de même pour CDRM, ce qui génère une augmentation importante du temps de réponse moyen. PEPR génère un temps de réponse important même si davantage de répliques sont créées. Cela confirme que des réplications répétées (par requête) génèrent un temps additionnel important alors que la plupart des réplications dans RSPC sont fournies par groupe de requêtes.

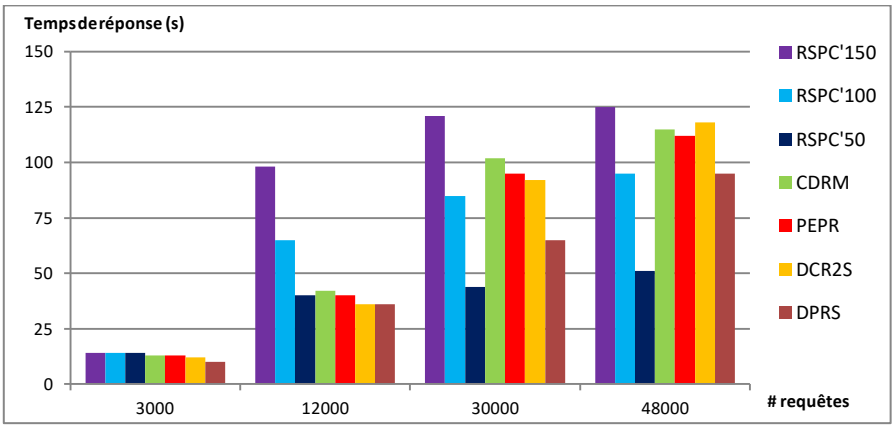

FIGURE 5.2 - Temps de réponse moyen avec une distribution zipf.

Nous mesurons également le temps de réponse et le facteur de réplication moyens lorsque les données sont distribuées de manière non uniforme, ici avec une distribution zipf. Dans ce cas, un plus grand nombre de répliques est créé pour les relations fréquemment utilisées. Le nombre de répliques est proportionnel à la popularité des données. Comme le montre la Figure 5.2, les temps de réponse de DCR2S, DPRS et RSPC sont inférieurs à ceux obtenus avec une distribution uniforme (Figure 5.1 (b)). DCR2S prend en compte la variation de la popularité des données en attribuant différentes pondérations aux données accédées à différents intervalles de temps. DPRS s'adapte dynamiquement aux préférences des utilisateurs tandis que RSPC maintient plus de répliques pour les données les plus populaires. Cela se traduit par une diminution du temps de réponse moyen (environ 11%, 9% et 7% avec DPRS, RSPC et DCR2S respectivement). D'un autre coté, aucune amélioration du temps de réponse n'a été constatée avec CDRM qui crée des répliques en se basant sur l'équilibrage de charge. C'est également le cas avec PEPR qui supprime périodiquement les répliques en fonction de la charge de travail, sans tenir compte de la popularité des données.

5.4.2 Ajustement du facteur de réplication

La Figure 5.3 montre le facteur de réplication moyen obtenu avec toutes les stratégies de réplication lorsqu'une diminution significative du nombre de requêtes est observée au cours d'une PF. Au cours de ces expériences, 24.000 requêtes ont été soumises lors de la première période (5 min) de PF (correspondant à 80 requêtes/s en moyenne), suivies de 1.500 requêtes au cours de la seconde période de PF (correspondant à 5 requêtes/s).

A la fin de la PF, le facteur de réplication diminue pour toutes les stratégies comparées. La réduction de la charge de travail implique une réduction de la consommation de ressources. Néanmoins, nous observons que RSPC supprime plus de répliques que CDRM, DCR2S et DPRS en raison de l'ajustement dynamique du facteur de réplication avec RSPC. Cela prouve

que CDRM, DCR2S et DPRS continuent à utiliser certaines des répliques précédemment créées même avec une charge de travail moins importante.

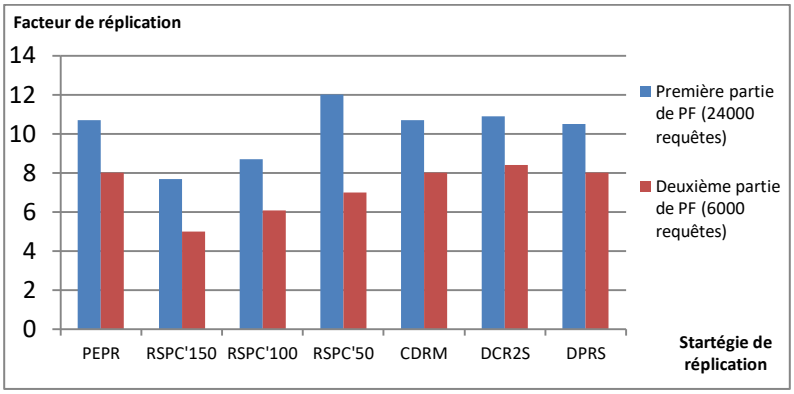

FIGURE 5.3 - Impact de la variation de la charge de travail sur le facteur de réplication.

Rappelons que les répliques inutiles deviennent inactives avec RSPC, c'est-à-dire que ces répliques ne seront pas accédées lors de l'exécution de futures requêtes pendant une certaine période (ici, 3 minutes). Elles seront, par contre, définitivement supprimées si la création d'une telle réplique n'est pas requise durant cette période.

5.4.3 Impact de la configuration du système sur le temps de réponse moyen

Dans cette section, nous avons choisi de varier la configuration du système afin d'observer le comportement de chaque stratégie. La Figure 5.4 (a) montre l'impact du nombre de MVs d'un CD sur les performances du système. Nous simulons la soumission de 30.000 requêtes de complexité aléatoire durant une PF.

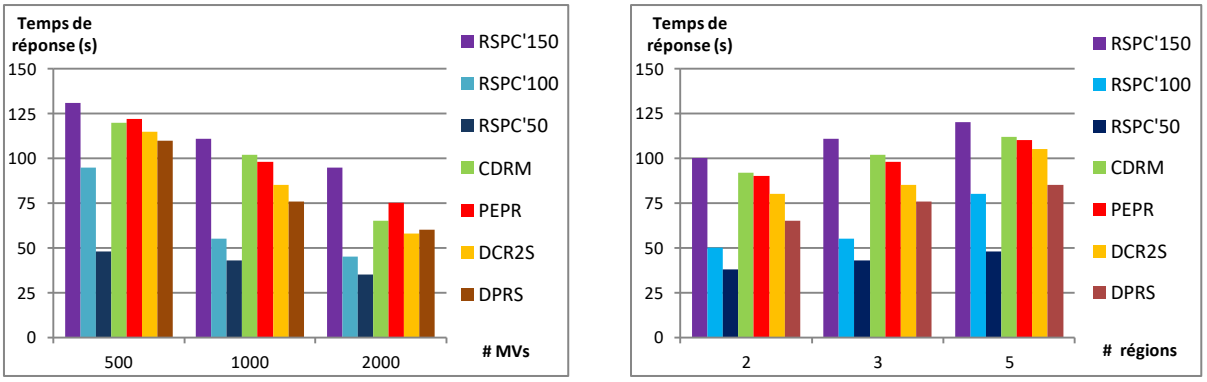

FIGURE 5.4 - Impact du (a) nombre de MVs et (b) du nombre de régions sur le temps de réponse.

À mesure que le nombre de MVs par CD augmente, le temps de réponse moyen diminue. Ceci est observé avec toutes les stratégies comparées. Cela est dû au fait que les MVs sont de moins en moins sollicitées au fur et à mesure qu'on en rajoute des MVs dans un CD. Entre 500 et 1 000 MVs par CD, le temps de réponse moyen diminue d'environ 20% avec CDRM et DCR2S et d'environ 25% avec DPRS. Cela est dû à la création supplémentaire de répliques au sein d'une région locale. Avec plus de MVs par CD (1500 MVs), DCR2S et CDRM ne créent

pas beaucoup d'autres répliques car le seuil budgétaire est déjà atteint avec DCR2S et l'équilibre de charge est déjà atteint avec CDRM. D'un autre coté, les données les plus populaires sont déjà répliquées avec DPRS bien que davantage de MVs soient sous-chargées. Le temps de réponse moyen obtenu avec RSPC'50 ne diminue que de 15% environ tandis que la diminution est d'environ 8% avec RSPC'150. Seules quelques répliques supplémentaires sont créées puisque SLO$_{RT}$ est satisfait. Cette réduction est encore moins importante si nous rajoutons 500 MVs supplémentaires.

L'expérience suivante consiste à varier plutôt le nombre de régions tout en maintenant le nombre total de MVs dans le système inchangé comme le montre la Figure 5.4 (b). Le but est d'observer l'impact du nombre de régions sur les performances du système. L'augmentation du nombre de régions engendre des temps de réponse moyens légèrement plus importants avec toutes les stratégies comme le montre la Figure 5.4 (b). Avec RSPC'50, moins de MVs sont disponibles pour créer d'autres répliques en exécutant le même nombre de requêtes. Rappelons qu'aucune réplique n'est créée en dehors de la région ayant reçu la requête avec RSPC. En revanche, plus de répliques sont créées en dehors de cette région avec CDRM, PEPR, DPRS et DCR2S, ce explique des temps de réponse moyens plus élevés.

5.4.4 Impact de la complexité des requêtes sur les performances

Il est intéressant de mesurer l'impact de la complexité des requêtes soumises par les locataires (en termes de nombre de jointures) sur la satisfaction du SLO$_{RT}$. Pour cela, nous avons simulé la soumission de 30.000 requêtes au cours d'une PF. Nous avons mesuré l'impact des requêtes simples, intermédiaires et complexes sur les performances. Dans la Figure 5.5, soumettre des requêtes simples signifie que 80% des requêtes sont simples, etc.

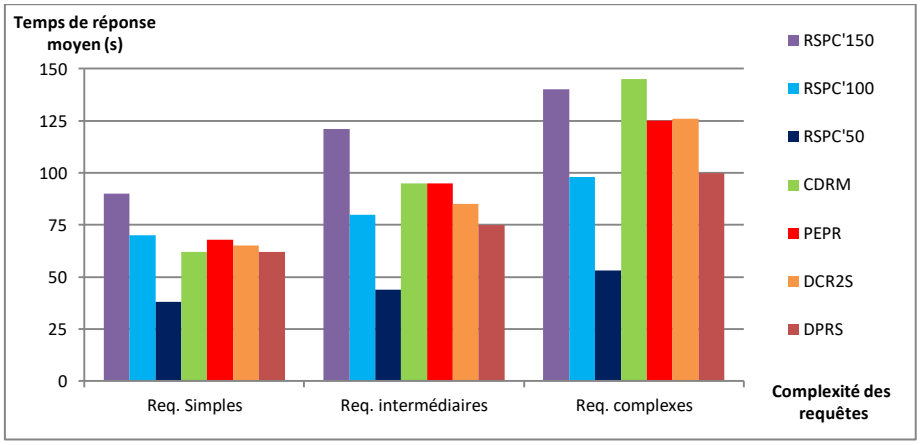

FIGURE 5.5 - Impact de la complexité des requêtes sur les performances.

Avec des requêtes simples, on obtient des temps de réponses presque similaires avec CDRM, DCR2S et DPRS lorsque RSPC'100 et RSPC'150 génèrent des temps de réponse plus importants comme le montre la Figure 5.5. RSPC'50 crée des répliques permettant au temps de réponse moyen d'être au dessous du seuil *RT_SLO_PSQ*. De leurs cotés, RSPC'150 et RSPC'100 fournissent des temps de réponses bien inférieurs aux *RT_SLO_PSQ* respectifs, ce qui ne nécessite pas la création de nouvelles répliques. Avec des requêtes dites intermédiaires, toutes les stratégies comparées fournissent des temps de réponses plus

importants. RSPC'50 crée davantage de répliques car le temps de réponse moyen n'est pas très éloigné du seuil *RT_SLO_PSQ*.

Avec des requêtes complexes, certaines MVs sont surchargées. En conséquence, nous observons des temps de réponse plus importants avec toutes les stratégies. Les MVs sont clairement bloquées pour la réception de nouvelles répliques dans une région locale avec CDRM. Des transferts de données entre régions sont alors nécessaires lors de la création de nouvelles répliques. Ceci est aussi le cas avec DPRS. La création d'autres répliques avec DCR2S n'est pas possible car le budget est atteint. On observe également des temps de réponses plus élevés avec PEPR à cause des réplications répétitives. D'autre part, le temps de réponse moyen obtenu avec RSPC'50 dépasse légèrement le seuil *RT_SLO_PSQ* alors que les temps de réponse moyens issus de PEPR, DCR2S et CDRM dépassent largement ce seuil. En revanche, les temps de réponse moyen obtenus avec RSPC100 et surtout de RSPC150 sont inférieurs aux seuils *RT_SLO_PSQ* respectifs.

5.4.5 Dépenses du fournisseur

Dans le chapitre précédent, nous avons vu qu'augmenter le profit économique du fournisseur revient à diminuer la consommation des ressources nécessaires à l'exécution des requêtes des locataires. Dans ce qui suit, nous mesurons les ressources consommées par le fournisseur lorsque différentes stratégies de réplication de données sont utilisées. Nous nous focalisons sur les ressources en termes de BP, de stockage et les pénalités payées aux locataires. Cela nous permet de déduire le profit économique perçu par le fournisseur au cours d'une PF lors de l'utilisation de chaque stratégie de réplication de données.

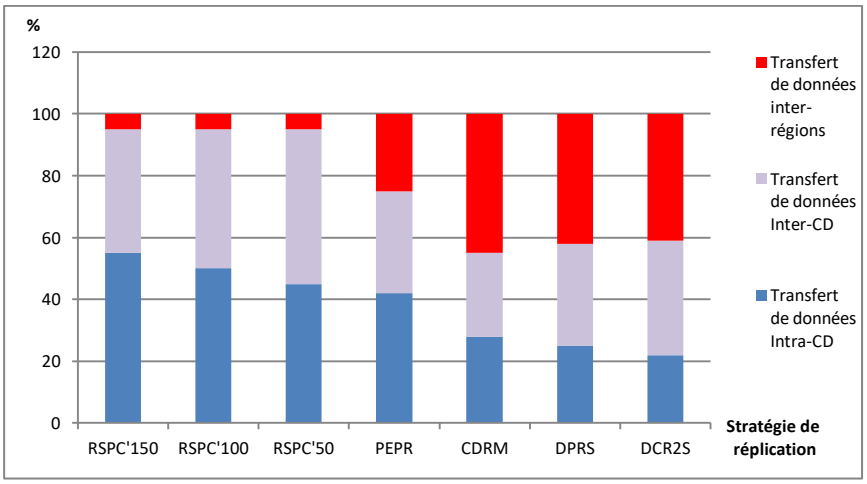

FIGURE 5.6 - Proportion des transferts de données par rapport à la hiérarchie de la bande passante du réseau.

5.4.5.1 Consommation en termes de Bande Passante (BP) du réseau

Le coût de transfert de données constitue une part importante des dépenses du fournisseur lors de l'exécution des requêtes des locataires. La Figure 5.6 montre la consommation du fournisseur en termes de BP par l'ensemble des stratégies comparées. Il est intéressant de mesurer la part de chaque type de transfert de données (intra-CD, intra-région et inter-région) d'autant plus que l'hétérogénéité concerne aussi bien les capacités en termes de BP que les coûts monétaires de transfert de données comme indiqué dans la Table 5.1.

Dans RSPC, les transferts de données sont essentiellement effectués au niveau intra-région. Les transferts de données entre régions sont effectués uniquement lors de la réplication visant à satisfaire l'objectif SLO$_{AV}$. En revanche, les transferts inter-régions sont plus fréquents avec DCR2S, DPRS, PEPR et surtout CDRM. Cette dernière stratégie place les nouvelles répliques en fonction de la charge des nœuds candidats et ne prend pas en compte la hiérarchie du point de vue de la BP. Cela a un impact négatif sur le temps de réponse moyen, car la BP disponible entre régions est largement inférieure à celle disponible au sein d'un même CD par exemple.

Afin de mesurer l'efficacité de l'utilisation de la BP pour chaque stratégie, la Formule (5.1) (Cameron et al., 2003) permet le calcul de l'efficacité d'utilisation du réseau *ENU* (*the Effective Network Usage*).

$$ENU = (\#\ dr + \#\ Repl)/\ \#dl \tag{5.1}$$

Dans cette formule, *#dr* indique le nombre de fois qu'un nœud (MV) accède à une relation depuis un nœud distant, *#Repl* correspond au nombre de réplications et *#dl* indique le nombre de fois qu'un nœud accède à une relation sur un disque local. La valeur de l'*ENU* est comprise entre 0 et 1. Une valeur *ENU* inférieure indique qu'une stratégie de réplication réussit à placer les données aux emplacements appropriés tout en permettant une utilisation plus efficace de la BP. La Figure 5.7 montre les valeurs *ENU* obtenues avec les stratégies comparées lorsque 30.000 et 48.000 requêtes sont soumises par PF.

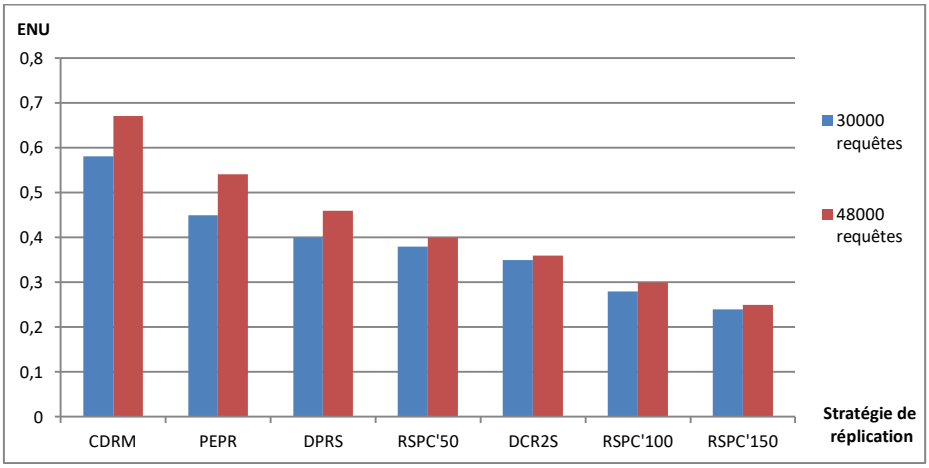

FIGURE 5.7 - Efficacité de l'utilisation du réseau (ENU) des stratégies comparées.

Comparée aux stratégies DPRS, PEPR et plus particulièrement à CDRM, la valeur *ENU* issue de RSPC'50 est plus petite (environ 5%, 8% et 20% respectivement). Cela est dû au fait que les données requises sont disponibles dans les régions locales avec RSPC. RSPC'100 et en particulier RSPSP'150 ne créent pas de beaucoup de nouvelles répliques vu que l'objectif SLO$_{RT}$ est presque tout le temps satisfait. En conséquence, les répliques existantes sont suffisantes pour la satisfaction de SLO$_{RT}$, ce qui génère moins de création de répliques et donc, moins de transfert de données. Par rapport au RSPC'50, la valeur *ENU* issue de DCR2S est inférieure de 3% car aucune réplique n'est créée lorsque le seuil du budget est atteint. En conséquence, seules certaines répliques sont crées localement. La simulation d'un plus grand nombre de requêtes (48.000 requêtes pendant une PF) engendre une augmentation de la

valeur *ENU* pour les stratégies CDRM, PEPR et DPRS (respectivement 9%, 8% et 5%), car davantage de répliques sont créées en dehors de la région qui reçoit la requête d'un locataire. En revanche, la valeur *ENU* issue de RSPC'150 n'augmente que de 2% vu que le nombre de répliques augmente lentement lors d'un pic de la charge de travail.

5.4.5.2 Consommation en termes de stockage de données

Nous mesurons également la consommation du fournisseur en termes de ressources de stockage pour l'ensemble des stratégies comparées. La consommation des ressources de stockage est étroitement liée au nombre de répliques crées par chaque stratégie. Evidemment, sans aucune réplication de données, la consommation des ressources de stockage correspond au stockage nécessaire pour l'hébergement des relations initialement placées. La Figure 5.8 montre le pourcentage de ressources de stockage utilisées dans le système.

Lorsque le fournisseur reçoit 12.000 requêtes, DCR2S, CDRM, PEPR et DPRS engendrent une consommation plus importante de ressources de stockage par rapport à RSPC car davantage de répliques sont créées. Dans ce cas, Il n'est pas nécessaire, la plupart du temps, de créer des répliques supplémentaires avec RSPC car le temps de réponse est inférieur au seuil *RT_SLO_PSQ*, en particulier avec RSPC'150. Avec 30000 requêtes, c'est RSPC'50 qui consomme le plus de ressources de stockage car davantage de répliques sont créées afin de satisfaire l'objectif SLO_{RT} alors que RSPC'150 nécessite moins de création de répliques vu que l'objectif SLO_{RT} est déjà satisfait.

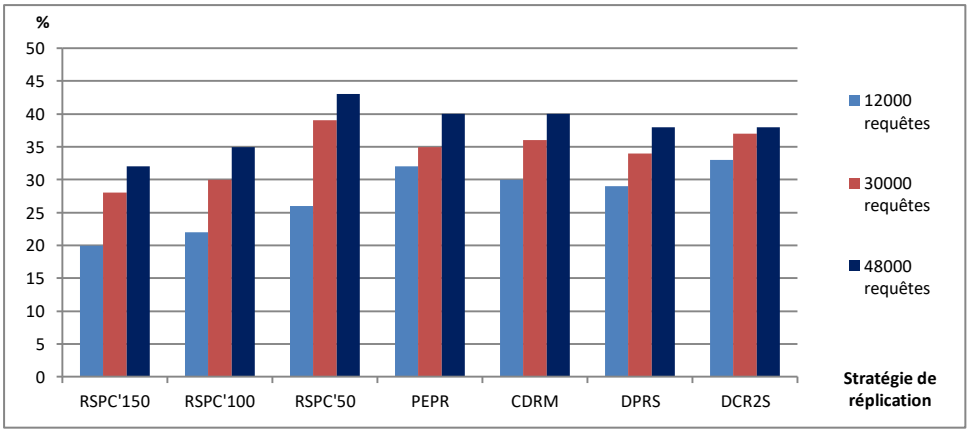

FIGURE 5.8 - Proportion des ressources de stockage utilisées par les stratégies comparées.

Un nombre plus important de requêtes engendre un nombre plus important de réplications. Pour cela, nous mesurons la consommation des ressources en termes de stockage avec un nombre plus important de requêtes. Lorsque 48.000 requêtes sont reçues, RSPC50 et DCR2S requièrent moins de 5% de ressources de stockage supplémentaires, car seules certaines répliques supplémentaires sont créées. Ceci n'est pas le cas des stratégies CDRM, PEPR et DPRS qui nécessitent des répliques supplémentaires afin de satisfaire l'objectif SLO_{RT}.

5.4.5.3 Analyse des violations du SLA

Le nombre de violations du SLA au cours d'une PF donne un bon aperçu sur l'efficacité d'une stratégie de réplication. L'analyse des violations du SLA peut être très utile notamment

lorsque le temps de réponse de certaines requêtes dépasse de loin le temps de réponse moyen. La Figure 5.9 montre le rapport entre le nombre de requêtes soumises par des locataires et le nombre de violations du SLA pendant une PF avec une distribution zipf de données. Nous supposons qu'une pénalité est payée avec PEPR, CDRM, DCR2S et DPRS lorsque le temps de réponse dépasse 100s.

Lorsque 12.000 requêtes sont soumises durant une PF, les stratégies comparées génèrent des nombres presque similaires de violations de SLA. En fait, la plupart des violations du SLA sont observées lors des premières requêtes dans la PF. Inévitablement, pendant une courte période au début des simulations, certaines violations sont simplement dues au rapatriement des données distantes afin de satisfaire SLO_{RT}. Le but est d'éviter ces violations de SLA à travers la réplication de données. Avec 30.000 requêtes, RSPC'100 et RSPC'150 génèrent moins de violations de SLA par rapport aux stratégies PEPR et CDRM. Le temps de réponse moyen obtenu avec CDRM et PEPR est proche de *RT_SLO_PSQ* mais la plupart des valeurs de temps de réponse dépassent ce seuil.

Avec un nombre élevé de requêtes (48.000 requêtes), PEPR, CDRM et DCR2S génèrent un plus grand nombre de violations du SLA. En revanche, ce nombre augmente lentement avec RSPC et DPRS. Ceci réduit le coût des pénalités, ce qui a un impact direct sur les dépenses du fournisseur. A titre d'exemple, le nombre de violations de SLA avec CDRM est 6 fois (2,2 fois respectivement) plus important que ceux générés par RSPC 150 (respectivement RSPC'50).

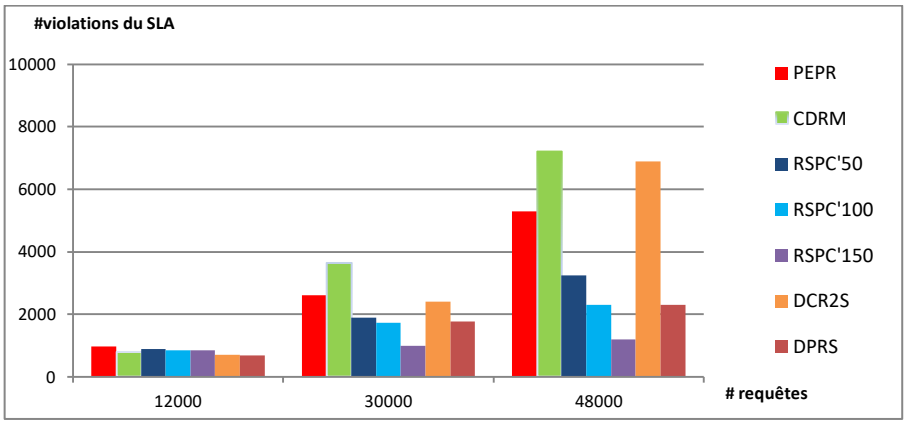

FIGURE 5.9 - Nombre de violations du SLA lors d'une période de facturation (PF).

5.4.5.4 Dépenses monétaires du fournisseur

La Figure 5.10 présente le total des dépenses monétaires du fournisseur accumulées au cours d'une PF en fonction des prix des ressources mentionnés dans la Table 5.1. Ces dépenses correspondent à la somme des coûts de ressources de calcul, de stockage, de réseau et des pénalités. Ces derniers coûts incluent les coûts nécessaires à l'exécution des requêtes mais également les coûts nécessaires à l'estimation du temps de réponse et du profit du fournisseur lors d'une réplication de données. Les coûts monétaires des pénalités dépendent du nombre de violation du SLA observées durant une PF.

Les locataires sont facturés pour l'exécution de 48.000 requêtes par le fournisseur au cours d'une PF. Avec les stratégies RSPC'50, RSPC'100 et RSPC'150, les revenus du fournisseur

correspondent respectivement à 48 $, 35 $ et 29 $, conformément aux prix d'exécution de 1000 requêtes, comme indiqué dans la Table 5.1. Les dépenses par requête sont calculées à partir de la Formule (4.13). Ensuite, le bénéfice du fournisseur est obtenu en soustrayant les dépenses des revenus, comme indiqué dans la Formule (4.1).

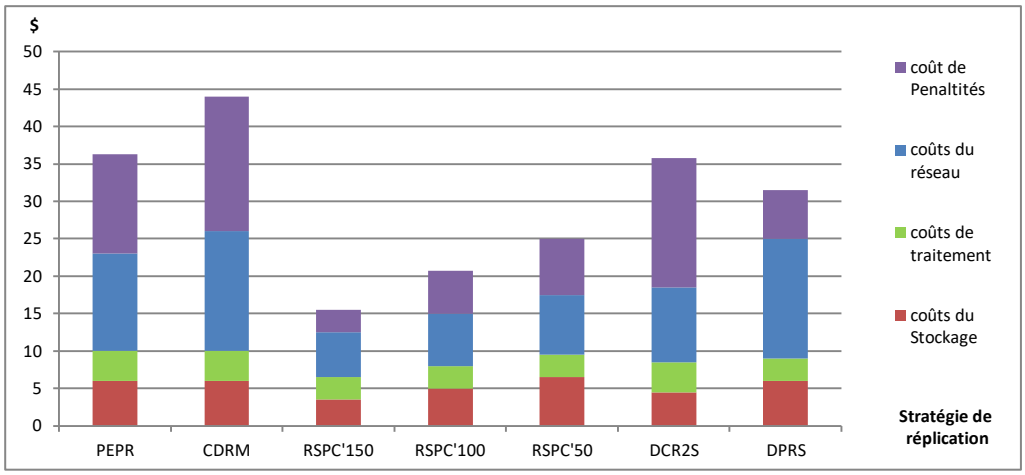

FIGURE 5.10 - Dépenses monétaires du fournisseur lors d'une période de facturation PF.

Du point de vue du coût monétaire des ressources de calcul, on observe presque le même coût pour le fournisseur avec l'ensemble des stratégies comparées. Par exemple, le coût engendré par les estimations du temps de réponse et du profit du fournisseur lorsqu'une réplication est envisagée avec RSPC est presque similaire à celui nécessaire au calcul de la probabilité de blocage sur une MV avec CDRM. Par ailleurs, nous avons vu que la stratégie RSPC génère moins de transferts de données car la majorité des répliques sont créées dans la même région. En conséquence, le fournisseur paie beaucoup moins de coûts liés au transfert de données d'autant plus que les coûts de transfert de données entre régions sont les plus élevés. D'un autre coté, RSPC'50 requiert davantage de ressources de stockage car davantage de répliques sont crées afin de satisfaire l'objectif SLO_{RT}. Cependant, la différence en termes de coûts monétaires n'est pas importante car le prix des ressources de stockage est relativement bon marché.

Beaucoup plus de pénalités sont payées avec les stratégies CDRM et DCR2S alors que la majorité des dépenses avec DPRS concerne le transfert de données. En revanche, moins de pénalités sont payées avec RSPC, en particulier avec RSPC 150, car le temps de réponse moyen ne dépasse pas le seuil *RT_SLO_PSQ*. En conséquence, les dépenses monétaires du fournisseur en utilisant RSPC sont inférieures aux dépenses générées par les autres stratégies. Par exemple, RSPC150 génère 3 fois moins de dépenses monétaires que CDRM. À revenus égaux, CDRM génère moins de profits pour le fournisseur que RSPC'100.

Si nous comparons maintenant les bénéfices générés par les différentes options de RSPC, nous déduisons de la Figure 5.10 que RSPC'50, RSPC'100 et RSPC'150 génèrent respectivement des bénéfices monétaires de 23 $, 15 $ et 12 $ respectivement. Ainsi, un nombre plus important de ressources allouées à un locataire génère plus de bénéfices pour le fournisseur. Par exemple, RSPC'50 génère plus de profits monétaires que RSPC'150. Cependant, moins de ressources seront disponibles pour les autres locataires avec RSPC'50.

En d'autres termes, plus de ressources seront encore disponibles avec RSPC'150, lors de l'exécution éventuelle d'autres requêtes.

5.5 ANALYSE DES RESULTATS

Avec un nombre réduit de requêtes, les temps de réponse moyens obtenus avec les stratégies PEPR, CDRM, DCR2S et DPRS sont légèrement meilleurs à ceux obtenus avec RSPC (les 3 options). Cela est dû à un plus grand nombre de répliques créées par CDRM, DCR2S et DPRS, qui ne prennent pas en compte les bénéfices du fournisseur. Cependant, avec un temps de réponse moyen légèrement supérieur tout en satisfaisant l'objectif SLO_{RT}, la stratégie RSPC génère moins de dépenses pour le fournisseur du Cloud.

À mesure que le nombre de requêtes augmente, les MVs deviennent plus occupées. Le même effet est observé avec des requêtes multi-jointures plus complexes. La stratégie RSPC envisage la création d'une réplique uniquement si l'objectif SLO_{RT} n'est pas satisfait. Ainsi, RSPC'50 génère un nombre important de créations de répliques alors que RSPC'150 ne se contente que de quelques créations de répliques. Avec CDRM, la satisfaction de l'équilibrage de charge n'est pas suffisante pour satisfaire SLO_{RT}. DPRS crée des répliques supplémentaires en fonction du changement de la popularité des données, alors que ces répliques sont créées en dehors de la région recevant les requêtes. En raison du seuil de budget fixé initialement avec DCR2S, la réplication n'est pas suffisante pour la satisfaction de SLO_{RT} ou encore pour l'équilibrage de charge. Le fait que RSPC'50 crée davantage de répliques, afin de satisfaire l'objectif SLO_{RT}, permet de réduire les coûts des pénalités payées au fournisseur. Ceci n'est pas le cas des stratégies CDRM, DCR2S et PEPR tandis que très peu de violations du SLA sont observées avec RSPC'150.

D'autre part, la réplication par requête dans PEPR génère des coûts supplémentaires, ce qui n'améliore pas forcément les performances du système. À l'opposé, la plupart des réplications sont effectuées par groupe de requêtes dans RSPC. En ce qui concerne la prise en compte des changements de comportement des utilisateurs, par exemple le mode d'accès aux données (très courant dans les environnements Cloud), les expériences ont montré que les stratégies RSPC, DCR2S et DPRS génèrent de meilleurs temps de réponse avec une distribution de données de type zipf.

Du point de vue de la consommation des ressources par le fournisseur, RSPC tire profit de la hiérarchie de la BP pour réduire la consommation de la BP du réseau alors que PEPR, DPRS et principalement CDRM nécessitent des transferts de données entre régions, bien plus chers que les transferts de données intra-région. D'un autre coté, bien que RSPC'50 nécessite plus de consommation de stockage lors de la création de nouvelles répliques, les coûts de stockage sont moins chers que ceux liés au transfert de données. En outre, plus de répliques inutiles sont supprimées avec RSPC grâce à l'ajustement dynamique du nombre de répliques. En conséquence, les coûts liés aux dépenses du fournisseur sont réduits, ce qui impacte directement les bénéfices du fournisseur.

Enfin, RSPC'150 génère des temps de réponse plus élevés que RSPC'50. Néanmoins, l'objectif SLO_{RT} est satisfait avec moins de ressources consommées. Le profit généré par RSPC'150 est également moins important pour le fournisseur. Néanmoins, davantage de ressources restent encore disponibles avec RSPC'150. Elles peuvent être utilisées pour

générer des bénéfices supplémentaires pour le fournisseur lorsque d'autres locataires seront servis.

5.6 CONCLUSION

Répondre au compromis entre les exigences de performances, en termes de temps de réponse, pour les locataires et le profit économique pour le fournisseur est un défi majeur pour les fournisseurs Cloud. La stratégie RSPC, évaluée tout au long de ce chapitre, ne vise pas à obtenir les meilleures performances comme c'est le cas dans les systèmes de grille de données. Elle vise à obtenir des performances acceptables pour les locataires tout en prenant en compte le profit du fournisseur. Ainsi, la réplication par requête ou en fonction uniquement de la popularité, ou encore la satisfaction d'un seul objectif tels que l'équilibrage de charge ne sont pas suffisants pour répondre à l'objectif des performances à moindre coûts pour le fournisseur.

L'évaluation des performances a montré que le taux d'arrivée des requêtes, la complexité des requêtes ou encore le mode d'accès aux données ont une influence significative sur la résolution du compromis cité plus haut. La réduction de l'espace de recherche, l'exploration du parallélisme ou encore l'ajustement dynamique du facteur de réplication sont quelques unes des solutions proposées. En comparant les stratégies RSPC et PEPR à trois autres stratégies proposées dans les systèmes Cloud, nous avons observé que RSPC répond mieux à l'objectif SLO_{RT} pour les locataires notamment avec des taux d'arrivée importants de requête, des seuils de temps de réponse stricts et des requêtes complexes. De plus, elle s'adapte mieux aux modifications du mode d'accès de l'utilisateur. En conséquence, les coûts de pénalités sont réduits avec RSPC. Les coûts de transfert de données sont également réduits en raison d'une réplication basée sur des transferts de données intra-région bien moins chers que les transferts de données inter-régions. Cela a un impact direct sur les bénéfices du fournisseur.

CHAPITRE 6

Conclusion et perspectives

Résumé

Ce chapitre conclut les travaux de recherche présentés tout au long de ce manuscrit. Nous terminons par développer quelques perspectives que nous envisageons de donner à ces recherches.

Sommaire

6.1 CONCLUSION

Les contributions présentées dans ce manuscrit s'inscrivent dans le contexte de la réplication de données dans les systèmes de gestion de données à grande échelle. Principalement, nous nous sommes focalisés sur la gestion de la réplication de données dans les systèmes de grille de données et dans les systèmes Cloud pour des applications décisionnelles.

Nous nous sommes d'abord intéressés à la réplication de données dans les systèmes de grille de données. Nous avons montré que les stratégies de réplication dans de tels systèmes doivent prendre en compte des caractéristiques telles que la grande échelle et la dynamicité des nœuds. Ces caractéristiques font que les stratégies de réplication de données proposées dans les systèmes distribués et parallèles classiques ne sont pas adaptées aux systèmes de grille de données. L'étude de l'état de l'art a montré que la plupart des travaux de synthèse existants ont classé ces stratégies en se basant sur des critères tels que la nature de la réplication (stratégies statiques vs. dynamiques), le mécanisme de contrôle (stratégies centralisées vs. décentralisées), ou encore l'origine de la réplication (stratégies initiées par le serveur vs. initiées par le client). Nous avons proposé, dans le chapitre 2, une nouvelle classification de ces stratégies basée sur deux autres critères : (i) la fonction objectif visée par une stratégie de réplication de données et (ii) le type d'architecture de la grille de données pour laquelle une stratégie a été conçue.

Nous avons analysé l'impact de ces critères sur les performances d'une stratégie de réplication. Une bonne stratégie doit trouver un compromis entre différents objectifs, parfois

conflictuels, afin d'améliorer les performances d'un système. Des facteurs tels que la consommation de la bande passante du réseau, le mode d'accès aux données ou encore la capacité de stockage ont un grand impact sur les performances d'une stratégie de réplication. D'un autre coté, une stratégie de réplication de données proposée pour des grilles de données avec une architecture hybride permet d'aboutir à de meilleures performances en tirant profit des avantages de plusieurs architectures, par exemple une bonne disponibilité à travers une architecture hiérarchique multi-niveaux et un passage à l'échelle à travers une architecture pair à pair. Par contre, une stratégie proposée pour une grille de données dont la topologie est à base de graphe, n'est pas disposée (bien que plus réaliste) à offrir les meilleures performances. En effet, la dynamicité des nœuds dans de telles architectures engendre des coûts élevés de maintenance de l'infrastructure.

Ensuite, nous nous sommes focalisés sur la réplication de données dans les systèmes Cloud. Créer autant de répliques que possible et viser les meilleures performances pour les utilisateurs, comme c'est le cas dans les systèmes de grille de données, n'est pas réaliste dans un environnement Cloud. Cela entraîne une consommation inutile de ressources et une réduction des profits pour le fournisseur. Des caractéristiques telles que la gestion élastique des ressources et la tarification de ces ressources pour les locataires suivant le modèle 'pay as you go' doivent aussi être prises en compte. Les stratégies de réplication de données dans les systèmes Cloud doivent alors répondre aux attentes des locataires en matière de performances tout en prenant en compte les coûts économiques engendrés par la réplication.

Nous avons aussi constaté que la plupart des travaux dans la littérature ont classé les stratégies de réplication proposées dans le Cloud comme étant statiques vs. dynamiques, et/ou centralisées vs. décentralisées. Nous avons proposé une autre classification basée sur d'autres critères, parfois spécifiques aux environnements Cloud : (i) l'orientation du bénéfice, (ii) la fonction objectif visée, (iii) le nombre d'objectifs visés, (iv) la nature de l'environnement pour lequel une stratégie a été conçue, notamment en ce qui concerne le nombre de fournisseurs, et (v) la prise en compte des coûts économiques lors de la réplication de données, avec un focus sur les stratégies qui prennent en compte le profit monétaire du fournisseur et la consommation énergétique lors de la réplication de données.

Aussi surprenant, la plupart des stratégies de réplication proposées dans la littérature ne s'intéressent qu'à la réduction de la consommation des ressources utilisées par le fournisseur lors de la réplication sans pour autant se focaliser sur les coûts économiques de cette réplication. De plus, de nombreuses stratégies se focalisent uniquement sur la satisfaction des objectifs de disponibilité de données ou de tolérance aux pannes. Il en est de même pour la plupart des fournisseurs commerciaux, qui n'incluent pas forcément l'objectif de temps de réponse pour les locataires dans le SLA. En effet, cet objectif est souvent conflictuel avec l'objectif d'obtenir un bénéfice économique maximal avec un coût d'exploitation minimal pour le fournisseur. Les stratégies de réplication des données dans de tels systèmes doivent alors prendre en compte le compromis entre la satisfaction de la QdS, en termes de temps de réponse, pour les locataires et la rentabilité économique pour le fournisseur, notamment lorsqu'elles sont proposées pour des applications décisionnelles.

Dans cette perspective, nous avons répondu aux problématiques liées à la réplication de données dans les systèmes Cloud composés de centres de données répartis à grande échelle. Concernant la prise de décision de la réplication, nous avons opté pour une approche basée sur des seuils. Le fournisseur propose à ses locataires un seuil de temps de réponse comme

garantie de performances au lieu d'une performance optimale. Nous avons proposé un modèle de coûts permettant l'estimation du temps de réponse d'un plan d'exécution parallèle d'une requête relationnelle, tout en exploitant les différents types et formes du parallélisme. Une création de réplique est envisagée seulement et seulement si l'objectif de temps de réponse n'est pas satisfait pour le locataire. De plus, cette création est considérée, le plus souvent, par groupe de requêtes afin de réduire les coûts liés à la réplication.

Concernant le placement d'une nouvelle réplique, nous nous sommes basés sur une heuristique qui exploite la hiérarchie au niveau de la bande passante du réseau, ce qui réduit sensiblement l'espace de recherche. Cette heuristique vise à trouver un placement acceptable qui satisfait le SLA tout en étant profitable pour le fournisseur. Pour cela, le modèle de coûts initial a été étendu afin d'intégrer les coûts économiques liés à l'exécution d'une requête. Les dépenses et les revenus monétaires du fournisseur sont estimés lorsqu'une réplication est envisagée. Les dépenses du fournisseur incluent non seulement le coût des ressources nécessaires à l'exécution d'une requête mais également, les coûts engendrés par la réplication ainsi que les coûts des pénalités éventuelles payées par le fournisseur à ses locataires en cas de violation du SLA. D'un autre coté, le fournisseur doit éviter de créer des répliques inutiles lorsque la charge de travail diminue. Nous avons alors proposé un algorithme d'ajustement dynamique du nombre de répliques permettant une gestion élastique des ressources. Cela permet au fournisseur d'adapter l'allocation des ressources aux requêtes des locataires. Enfin, nous avons proposé un algorithme qui évite le payement répétitif de pénalités par le fournisseur à ses locataires. Ses dépenses sont alors réduites.

Nos propositions ont été implémentées et validées en comparant leurs performances à celles d'autres stratégies proposées dans le Cloud. L'analyse des résultats, obtenus par simulation, a montré que nos propositions ne se contentent pas seulement de satisfaire les objectifs des locataires. Elles prennent aussi en compte le profit économique du fournisseur. La réduction de l'espace de recherche réduit sensiblement les coûts liés aux transferts de données et la réplication par groupe de requêtes diminue les dépenses liées à la réplication, ce qui a un impact réel sur le profit du fournisseur. De plus, nos propositions permettent d'obtenir de meilleures performances dans des conditions défavorables, par exemple des taux importants d'arrivée de requêtes, des seuils de temps de réponse stricts et des requêtes complexes.

6.2 PERSPECTIVES

Dans la continuité des travaux de recherche présentés tout au long de ce manuscrit, les suites que nous envisageons de donner à nos activités de recherche sont nombreuses. Ci dessous, nous présentons quelques nouvelles directions :

6.2.1 Réplication de données dans les systèmes Cloud multi-fournisseurs

Les stratégies issues de nos travaux de recherche et présentées dans le chapitre 4 ont été proposées dans un système Cloud mono-fournisseur. Dans la plupart de nos contributions, on ne cherche pas à trouver les ressources les moins chères lors de la réplication de données. Seule l'hétérogénéité des prix des ressources au sein des différents centres de données est prise en compte lors de l'estimation des dépenses du fournisseur. En d'autres termes, la différence de prix entre différents CDs n'est pas considérée comme un critère de décision lors de la réplication.

Dans la littérature, seules quelques stratégies de réplication de données ont été proposées pour un environnement Cloud multi-fournisseurs (Wu et al., 2013; Liu and Shen, 2017; Mansouri and Buyya, 2019). Néanmoins, un récent sondage réalisé par Kentik montre que de plus en plus de clients du Cloud public utilisent souvent plus d'un fournisseur. Amazon Web Services et Microsoft Azure sont le plus souvent choisis par des clients qui visent des services variés en fonction des politiques de tarification appliquées par tel ou tel fournisseur. Ces fournisseurs de services de Cloud publics sont souvent liés à des centres de données existants et répartis à travers le monde. Dans ce contexte, nous prévoyons d'étendre nos propositions afin qu'elles soient opérationnelles dans un tel environnement.

D'un coté, un fournisseur pourra exploiter les prix des ressources disponibles dans d'autres fournisseurs afin de réduire ses coûts liés à la réplication. Cela nécessite l'extension de notre modèle de coûts économiques avec des répercussions sur l'heuristique de placement de répliques. Dans ce cas, il s'agit d'une réplication de données qui vise l'augmentation du profit du fournisseur. D'un autre coté, une stratégie de réplication de données peut également exploiter cette hétérogénéité des prix de ressources afin de réduire les dépenses des locataires, par exemple en termes de stockage et d'énergie. Dans ce cas, on parle alors de stratégies de réplication de données orientées locataires. Dans les travaux proposés par (Yin et al., 2018), un locataire fournit seulement quelques statistiques au fournisseur lors de la négociation du SLA. Par la suite, les offres de différents fournisseurs sont comparées de telle sorte que ce locataire puisse choisir la meilleure offre afin de réduire ses dépenses.

6.2.2 Exploitation de la technique des codes correcteurs pour réduire les dépenses du fournisseur

La technique des codes correcteurs (Sathiamoorthy et al., 2013), largement connue dans les systèmes P2P, est utilisée par de nombreux Cloud commerciaux tels que Facebook, Microsoft et Google comme une alternative ou en appui à la réplication pour économiser l'espace de stockage tout en assurant une tolérance aux pannes optimale. Dans une telle technique, des données redondantes sont codées et stockées dans différents emplacements. Puis, elles peuvent être régénérées à partir d'un sous ensemble de données.

Il est évident que le codage/décodage des données peut nécessiter des ressources de calcul supplémentaires. Pour cela, la stratégie de réplication de données HyRD proposée dans (Mao et al., 2016) s'appuie sur la réplication de données pour stocker de petits fichiers et sur la technique de codes correcteurs pour stocker des fichiers volumineux. Dans nos travaux visant l'amélioration des performances des applications OLAP, nous pourrons envisager de répliquer les données populaires, dites données chaudes, et utiliser les codes correcteurs pour les données moins populaires, dites données froides (Bui et al., 2016).

D'autres techniques telles que la compression (Liu and Shen, 2016) ou la déduplication (Xu et al., 2015a) de données permettent également de réduire la consommation de ressources pour le fournisseur. En effet, la réplication de données volumineuses peut engendrer une consommation importante des ressources. La compression de données permet alors de réduire la consommation de ressources en termes de bande passante du réseau lors du transfert de données et en termes de stockage. La déduplication permet aussi de réduire les données transférées sur le réseau. Elle consiste à fragmenter une relation puis, d'associer un identifiant unique pour chaque fragment. Un même fragment n'est stocké qu'une seule fois.

Ensuite, seuls les fragments non présents dans un nœud de placement sont transférés à travers le réseau.

6.2.3 Maximisation du profit du fournisseur tout en assurant une QdS pour les locataires

Les stratégies de réplication proposées tout au long de ce manuscrit visent à prendre en compte le compromis entre la satisfaction des objectifs pour les locataires et la rentabilité pour le fournisseur. Il est évident que le fait qu'un fournisseur puisse servir un plus grand nombre de locataires, lui permet d'augmenter son profit économique, même si cela engendre une diminution des performances pour ses locataires (à condition qu'elles restent satisfaisantes). En se basant sur l'isolation des performances et la réplication de données, nous pourrons envisager de répondre aux exigences d'un nombre optimal de locataires afin de maximiser le profit du fournisseur.

Dans cette perspective, il faudrait d'abord trouver le nombre optimal de locataires tels que le bénéfice du fournisseur soit maximal. Afin d'éviter l'interférence entre les différents locataires servis, l'isolation des performances (Lama et al., 2018) permet de fixer dynamiquement la quantité absolue des ressources requises par locataire (par exemple, en termes de CPU, E/S, mémoire et réseau) tout en précisant l'objectif des performances et les pénalités éventuelles dans le SLA. Ensuite, une création d'une réplique sera envisagée, par groupe de requêtes et par locataire, à chaque fois que l'exigence des performances n'est pas satisfaite pour un locataire, suivant ce qui a été fixé comme ressources pour ce locataire.

L'isolation des performances présente quelques inconvénients tels que la nécessité pour un locataire de connaitre à l'avance les ressources nécessaires pour l'exécution de ses requêtes. Dans nos travaux futurs, on pourra se baser sur les travaux de (Ortiz et al., 2015) permettant à un locataire de spécifier uniquement le schéma de la base de données avec quelques statistiques de base comme les cardinalités des relations de base. Cela permet à un fournisseur de mieux prévoir les ressources requises par chaque locataire.

De plus, un locataire peut avoir ses propres objectifs SLO, par exemple un seuil de temps de réponse personnalisé. On parle alors d'un SLA personnalisé. Dans ce cas, une extension de notre modèle de coûts économique est envisageable afin de prendre en compte l'hétérogénéité des accords de SLA. Dans les travaux de (Ortiz et al., 2015), le fournisseur de Cloud affiche différents niveaux de services qui peuvent être fournis pour différent modèles de requêtes d'un locataire. Chaque niveau de service contient les performances estimées et le prix correspondant pour chaque modèle. On pourra également envisager l'introduction, par le fournisseur, d'un poids pour chaque locataire. Ensuite, le fournisseur pourra privilégier la satisfaction d'un locataire avec lequel un plus grand bénéfice est attendu au dépend d'autres locataires moins rentables.

6.2.4 Réplication de données basée sur l'apprentissage par renforcement

Afin d'assurer le dimensionnement automatique des ressources, nous avons opté pour une élasticité horizontale à travers la réplication de données basée sur des seuils. Rappelons qu'un seuil de temps de réponse, intégré dans le SLA, est préalablement négocié entre le fournisseur et ses locataires.

De nombreux fournisseurs de Cloud se basent sur l'approche de seuils à cause de sa nature intuitive. Néanmoins, la définition des seuils de manière efficace et le choix des paramètres à prendre en compte exigent une connaissance approfondie des tendances de la charge de

travail ainsi qu'une intervention humaine afin de fixer le seuil pour chaque métrique. Dans ce contexte, certains travaux se basent sur la prédiction : (i) des seuils en utilisant des séries chronologiques (Khatua et al., 2010), (ii) du facteur de réplication (Ananthanarayanan et al., 2011) ou (ii) de la charge de travail (Calheiros et al., 2015). D'autres travaux (Ghanbari et al., 2011) combinent l'approche des seuils avec la théorie de contrôle permettant l'obtention de seuils dynamiques en se basant sur une modélisation mathématique de la charge de travail.

Afin d'éviter l'intervention humaine lors de la définition des seuils, nous pourrons considérer une réplication de données basée sur l'apprentissage par renforcement (Ferreira et al., 2020). Dans les algorithmes d'apprentissage par renforcement tel que le Q-learning, un agent autonome dispose d'un certain nombre d'actions possibles permettant le changement de l'état d'un environnement. Il reçoit alors une récompense (ou une pénalité) pour chacune de ses actions. Ensuite, cet agent doit mémoriser la séquence des actions qui maximise sa récompense totale. Néanmoins, cette approche nécessite une période d'apprentissage.

Seuls quelques travaux de dimensionnement automatique basés sur l'apprentissage par renforcement dans le Cloud sont dédiés à l'interrogation de bases de données relationnelles. La plupart se sont intéressé aux systèmes NoSQL (Huang et al., 2013; Naskos et al., 2018). Les méthodes existantes doivent alors être adaptées au contexte des bases de données relationnelles avec notamment, la prise en compte de nombreuses tâches dépendantes et des relations intermédiaires qui peuvent être stockées sur le disque. Dans ce contexte, un agent peut mémoriser certaines actions lui permettant de privilégier la création rentable d'une réplique d'une relation, tout en satisfaisant les objectifs des locataires. L'efficacité des agents peut également être améliorée en s'appuyant sur un journal de requêtes afin de connaitre les requêtes les plus fréquentes ainsi que les données les plus accédées. Un agent pourra alors prévoir les périodes à forte charge de travail et les données qui seront les plus populaires dans le futur. En conséquence, des ressources peuvent être allouées à l'avance.

6.2.5 Animation de la recherche et feuille de route en lien avec mes perspectives

Depuis mon recrutement, j'ai eu la chance de travailler dans une petite équipe (3 puis 4 titulaires depuis 2012 et 3 doctorants en moyenne), ce qui m'a permis d'éviter l'éparpillement thématique. Bien sûr, cela réduit les capacités de l'équipe de répondre à de nombreux appels à projet, mais nous a permis en revanche, de se concentrer sur la recherche fondamentale. Cela ne nous a pas empêchés de participer à quelques projets (voir Annexe A) et de lancer des collaborations scientifiques au niveau local (co-encadrement d'une thèse avec une autre équipe à l'IRIT), national (participation à un projet ANR entre 2009 et 2013), industriel (lancement d'une thèse CIFRE en octobre 2020) et international (deux projets PHC, en 2010 et en 2020).

En plus de ces collaborations, le fait d'exposer mes travaux de recherche dans le cadre de séminaires de recherche et d'un programme de mobilité scientifique (Projet TOR – Suède, en 2018) à l'étranger, et de s'investir dans l'encadrement des stages m'ont permis par la suite d'encadrer des thèses soit en co-tutelle ou à travers d'autres types de collaborations officielles (voir Annexe A). Enfin, mes responsabilités pédagogiques (notamment la responsabilité au sein du parcours math-info SID) m'ont permis d'avoir une vision interdisciplinaire et d'acquérir quelques connaissances en statistiques (à travers les discussions avec mes collègues mathématiciens), qui m'ont été très utiles lors des différentes expérimentations permettant la validation de mes résultats de recherche théoriques.

A coté des activités récurrentes d'enseignements, de responsabilités pédagogiques et d'animation de la recherche, mes travaux de recherche sont centrés autour de l'évaluation et l'optimisation des requêtes réparties dans des environnements à grande échelle. Au cours des prochaines années et dans la continuité de ces travaux de recherche, j'envisage de poursuivre dans la recherche fondamentale en visant la conception de nouveaux modèles d'allocation de ressources dans les systèmes de gestion de données à grande échelle avec comme objectif le compromis (encore) entre la recherche fondamentale et l'application de ces recherches via des projets de recherche collaborative appliquée. Ainsi, mon objectif est de concrétiser les perspectives présentées dans ce chapitre à travers le lancement de quelques thèses académiques ainsi que le montage de projets avec des partenaires industriels.

La première perspective visant la conception de stratégies de réplication de données pour des systèmes Cloud multi-fournisseurs constitue une bonne opportunité pour lancer une thèse, d'autant plus que les locataires optent de plus en plus pour plusieurs fournisseurs suivant le service demandé. En effet, la tarification des ressources peut varier en fonction du fournisseur et de son implantation dans une région du monde.

Ensuite, la perspective relative à la maximisation du profit du fournisseur tout en assurant une QdS pour des millions de locataires est un défi majeur pour les fournisseurs Cloud. Malgré le fait que l'offre de services en lien avec l'Internet des objets et l'intelligence artificielle a ouvert le champ à de nouveaux acteurs tels qu'IBM et Alibaba Cloud, les principaux fournisseurs de services (Amazon Web Services, Microsoft Azure et Google Cloud platform) continuent d'accentuer leur avance et dominent largement le marché mondial. En conséquence, une poignée de fournisseurs se livrent une concurrence acharnée afin d'attirer le plus grand nombre possible de locataires. Dans cette perspective, concevoir des stratégies de réplication permettant de maximiser le profit du fournisseur tout en satisfaisant les exigences d'un nombre optimal de locataires pourrait faire l'objet d'une thèse. Dans ce contexte, la combinaison de la réplication de données et l'isolation des performances, et l'extension de notre modèle économique afin de prendre en compte l'hétérogénéité des accords SLA constituent quelques unes des pistes envisagées.

Enfin, une réplication de données basée sur l'apprentissage par renforcement permet à un fournisseur d'éviter toute intervention humaine, nécessaire par exemple lors de la configuration du dimensionnement automatique basé sur des seuils. Cela peut constituer un sujet de thèse attractif, notamment pour des partenaires industriels qui pourraient être intéressés par le montage d'une thèse CIIFRE. Il en est de même pour la perspective de combiner les techniques de la réplication de données et des codes correcteurs afin de réduire la consommation des ressources allouées aux locataires, ce qui permet d'augmenter le profit du fournisseur.

BIBLIOGRAPHIE

L. Abad, Y. Lu, R. Campbell, DARE: Adaptive Data Replication for Efficient Cluster Scheduling, Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER 11), pp. 159-168, (2011).

D. J. Abadi. Data management in the cloud : Limitations and Oppertunities. IEEE Bulletin of the Technical Commmittee on Data Engineering, Vol. 32, Issue 1, pp. 3-12, (2009).

A. Abdullah, M. Othman, H. Ibrahim, M.N. Sulaiman, A.T. Othman. Decentralized replication strategies for P2P based scientific data grid, in: Int. Symposium on Information Technology (ITSim), Vol. 3, pp. 1–8, (2008).

A. Abouzamazem and P. Ezhilchelvan. Efficient inter-Cloud réplication for high-availability services. Int. Conf. on Cloud Engineering (IC2E), pp. 132–139. IEEE, (2013).

H. Abu-Libdeh, L. Princehouse and H. Weatherspoon. Racs: A case for cloud storage diversity. In ACM Workshop on Cloud Computing (SoCC) in conjunction with SIGMOD, Indianapolis, USA, pp. 229-240, (2010).

A. R Abdurrab and T. Xie. FIRE: A file reunion based data replication strategy for data grids. In 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, IEEE, pp. 215–223. (2010)

D. Agrawal, A. E., Abbadi, F. Emekci, A. Metwally. Database management as a service: Challenges and opportunities. International conference on Data Engineering. pp. 1709–1716. (2009)

Y. Al-Dhuraibi, F. Paraiso, N. Djarallah, P. Merle. Elasticity in cloud computing : state of the art and research challenges. IEEE Transactions on Services Computing, 11(2), pp. 430–447. (2017)

M. Alghamdi, B. Tang, Y. Chen. Profit-based file réplication in data intensive Clouddata centers. IEEE International Conference on Communications (ICC), Paris, France, pp. 1–7, (2017).

H. E. Al Mistarihi and C. Yong. Replica Management in Data Grid. Proc. of Int. Journal of Computer Science and Network Security, Vol. 8, pp. 22–32, (2008).

I. Al Ridhawi, N. Mostafa and W. Masri. Location-aware data replication in cloud computing systems. Int. Conf. on Wireless and Mobile Computing, Networking and Communications (WiMob), IEEE, pp. 20–27, (2015)

M. Ali, B. Kashif, U. Khan, V. Bhardwaj, L. Keqin, Z. Albert. DROPS: Division and Replication of Data in Cloud for Optimal Performance and Security. IEEE transaction on cloud Computing, pp. 303-315, (2018)

T. Amjad, M. Sher and A. Daud. A survey of dynamic replication strategies for improving data availability in data grids. Future Generation Computer Systems, 28(2): pp. 337–349, (2012)

G. Ananthanarayanan, S. Agarwal, S. Kandula, A. Greenberg, I. Stoica, D. Harlan, E. Harris. Scarlett: Coping with Skewed Content Popularity in MapReduce Clusters. Int. Conf. on Comp. System (EuroSys), pp. 287-300, (2011)

V. Andronikou, K. Mamouras, K. Tserpes, D. Kyriazis, T. Varvarigou, Dynamic QoS-aware data replication in grid environments based on data 'importance', Future Generation Computer Systems (28). pp. 544-553, (2012)

M.S. Ardekani, D. B. Terry. A self-configurable geo-replicated cloud storage system. 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI), Broomfield, pp. 367- 381, (2014).

M. Arlitt, L. Cherkasova, J. Dilley, R. Friedrich, T. Jin. Evaluating content management techniques for Web proxy caches. ACM SIGMETRICS Performance Evaluation Review, 27(4): pp. 3–11, (2000).

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

AWS Educate. https://aws.amazon.com/fr/education/awseducate/ (décembre 2020)

L. Azari, A.M. Rahmani, H.A. Daniel, N.N. Qader. A data replication algorithm for groups of files in data grids. J. of Parallel Distributed Computing (JPDC). pp. 115–126, (2018)

KK. Azumah. Hybrid Cloud Adoption in a Developing Economy: An architectural overview. Handbook on Ict in Developing Countries, (2019)

X. Bai, H. Jin, X. Liao, X. Shi, and Z. Shao. RTRM: A response time-based replica management strategy for cloud storage system. Grid and Pervasive Computing, N. 1, pp. 124–133. (2013).

L. A. Barroso, U. Holzle, P. Ranganathan. The Datacenter as a Computer: Designing Warehouse-Scale Machines. 3rd Edition. Synthesis Lectures on Computer Architecture. 189 pages, (2018)

S. S. Begum and S. Sirisha. Cloud optimal security using data fragmentation and replication. Int. J. Of Computer Science Ingeneering and scientific technology (IJCSEST). ISSN 6201 3454, (2019)

M. Beigrezaei, A. Toroghi Haghighat, M. Reza Meybodi and M. Runiassy. PPRA: A new pre-fetching and prediction based replication algorithm in data grid. In 6th International Conference on Computer and Knowledge Engineering (ICCKE), number ICCKE, IEEE, pp. 257–262, (2016)

G. Belalem, S. Bouamama, L. Sakhri. An Effective Economic Management of Resources in Cloud Computing. Journal of Computers, Vol. 6, N. 3, pp. 404-411, (2011)

W. H. Bell, D. G. Cameron, R. Carvajal-Schiaffino, A. Paul Millar, K. Stockinger and F. Zini. Evaluation of an economy-based file replication strategy for a data grid. Proc. of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID), pp. 661–668, (2003).

W. H. Bell, D. Cameron, A. Millar, L. Capozza, K. Stockinger, F. Zini. Optorsim: A Grid Simulator for Studying Dynamic Data Replication Strategies. J. of High Performance Computing Applications, 17(4), pp. 403-416, (2003).

A. Benoit, V. Rehn-Sonigo, Replica Placement and Access Policies in Tree Networks. IEEE Transactions on Parallel and Distributed Systems. Vol. 19, Issue 12, pp. 1614-1627, (2008).

D. Benslimane, M. Barhamgi, F. Cuppens, F. Morvan, B. Defude, E. Nageba, F. Paulus, S. Morucci, M. Mrissa, N. Cuppens-Boulahia, C. Ghedira, R. Mokadem, S. Oulmakhzoune, J. Fayn. A Privacy-Preserving Service-oriented Data Integration System. SIGMOD Record, ACM Press, Vol. 42 N. 3, p. 42-47, (2013).

P. A. Bernstein, V. Hadzilacos, N. Goodman. Concurrency Control and Recovery in Database Systems. Massachusetts, Addision-Wesley Publishers, Book ISBN 0-201-10715-5, (1987).

A. Bessani, M. Correia, B. Quaresma, F. André and P. Sousa, DepSky: dependable and secure storage in a cloud-of-clouds, EuroSys'11:Proc. 6thConf. On Computer systems: pp. 31-46 (2011).

A. N. Bessani, M. Correia, B. Quaresma, F. André, and P. Sousa. DepSky: Dependable and secure storage in a cloud-of-clouds. ACM Trans. Storage, Vol. 9, N. 4, paper number 12, (2013).

R. Bolze, F. Cappello, E. Caron et al. Grid'5000: a large scale and highly reconfigurable experimental Grid testbed.Int.l J.l of High Performance Computing Applications, 20(4): pp. 481–494, November (2006).

N Bonvin, T. G Papaioannou, K. Aberer. A self-organized, fault tolerant & scalable replication scheme for cloud storage categories and subject descriptors. First ACM symposium on Cloud computing, pp. 205–216, (2010a).

N. Bonvin, T. G. Papaioannou, and K. Aberer. An economic approach for scalable and highly-available distributed applications. IEEE 3rd International Conference on Cloud Computing (CLOUD), pp. 498–505, (2010b).

N. Bonvin, T. G. Papaioannou, K. Aberer. Autonomic SLA-driven Provisioning for Cloud Applications. In 11th Int. Symp. on Cluster, Cloud and Grid Computing, pp. 434- 443, (2011).

D. Boru, D. Kliazovich, F. Granelli, P. Bouvry and A. Y. Zomaya. Energy-efficient data replication in cloud computing datacenters. Cluster Computing, 18(1), pp. 385-402, (2015).

L. Bouganim. Exécution parallèle de requêtes relationnelles et équilibrage de charge. Projet Rodin – INRIA Rocquencourt, (1997)

L. Breslau , P. Cao , L. Fan , G. Phillips , S. Shenker , Web caching and Zipf-like distributions: evidence and implications, in: Proc. of IEEE INFOCOM'99, New York, USA, pp. 126–134, (1999)

E. A. Brewer. Towards robust distributed systems (abstract). In Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing, PODC '00, New York, NY, USA, page 7, (2000)

M. Bsoul, A. Al-Khasawneh, E. Abdallah and Y. Kilani. Enhanced fast spread replication strategy for data grid. Journal of Network and Computer Applications, 34(2): pp. 575–580. (2011).

D.M. Bui, S. Hussain, E. N. Huh, S. Lee. Adaptive replication management in HDFS based on supervised learning. IEEE Transactions on Knowledge and Data Engineering, Vol. 28, Issue 6, pp.1369-1382, (2016)

R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, I. Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation Computer Systems, (25)17, pp. 599-616, (2009)

R.N. Calheiros, R. Ranjan, A. Beloglazov, C. De Rose, R. Buyya. CloudSim: a Toolkit for Modeling and Simulation of Cloud Computing Env. & Evaluation of Resource Provisioning Algorithms. Software: Practice & Experience. Vol. 41, Issue 1, pp. 23-50. (2010)

R. N. Calheiros, E. Masoumi, R. Ranjan, and R. Buyya. Workload Prediction Using ARIMA Model and Its Impact on Cloud Applications' QoS. IEEE Transactions on Cloud Computing 3, 4, pp. 449–458, (2015).

D.G. Cameron, R. Carvajal-schiaffino, A. Paul Millar, C. Nicholson, K. Stockinger, F. Zini. UK grid simulation with OptorSim. In e-science all-hands meeting, Nottingham, UK, (2003)

D. G. Cameron, R. Carvajal-Schiaffino, A.P. Millar, C. Nicholson, K. Stockinger, F. Zini, OptorSim: a grid simulator for replica optimisation, in: UK e-Science all Hands Conference, Vol. 31, (2004).

R. A. Campêlo, M. A. Casanova, D. O. Guedes et al. A brief survey on replica consistency in cloud environments. J. of Internet Services and Applications. Vol. 11, N. 1 (2020)

M. Carman, F. Zini, L. Serafini, K. Stockinger. Toward an Economy-Based Optimization of File Access and Replication on Data Grid. The 2nd Int. Symposium on Cluster Computing and the Grid. pp 340-345, (2002)

Casas, J. Taheri, R. Ranjan, L. Wang, A. Y. Zomaya, A balanced scheduler with data reuse and replication for scientific workflows in cloud computing. Fut. Gener. Comput. Syst. Vol. 74, pp. 168–178, (2017)

Z. Challal, T. Bouabana-Tebibel. A priori replica placement strategy in data grid. IEEE Int. Conference on Machine and Web Intelligence, pp. 402–406. (2010)

V. Chang. Towards a big data system disaster recovery in a private cloud. Ad Hoc Networks, Vol. 35 : pp. 65–82. (2015)

R.S. Chang, P.H. Chen. Complete and fragmented replica selection and retrieval in Data Grids. Future Generation Computer Systems. 23(4), pp. 536–546, (2007)

R.S. Chang, H.P. Chang, A dynamic data replication strategy using access weights in data grids, The Journal of Supercomputing 45 (3) pp. 277–295, (2008).

S. Chaudhuri. What next?: a half-dozen data management research goals for big data and the cloud. The 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2012, Scottsdale, AZ, USA, pp. 1–4, (2012)

D. Chen, S. Zhou, X. Ren and Q. Kong. Method for replica creationin data grids based on complex networks. J. of China Universities of Posts and Telecommunications, 17(4): pp. 110–115, (2010)

H. Chen, Y. Hu, P. Lee, and Y. Tang. NCCloud: A network coding- based storage system in a cloud-of-clouds. IEEE Trans. Comput., vol. 63, no. 1, pp. 31–44, (2014)

Z. Cheng, Z. Luan, Y. Meng, Y. Xu, D. Qian, A. Roy, N. Zhang, G. Guan. ERMS: An Elastic Replication Management System for HDFS. IEEE Int. Conf. on Cluster Computing Workshops (CLUSTER), pp. 32-40, (2012)

A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, S. Tuecke. The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets. J. of Network and Computer Applications, 23(3): pp. 187–200, (2000)

A. Chervenak, E. Deelman, I. Foster, L. Guy, W. Hoschek, A. Iamnitchi, C. Kesselman, P. Kunszt, M. Ripeanu, B. Schwartzkopf. Giggle: A Framework for Constructing Scalable Replica Location Services. Proc. of the ACM/IEEE Conference on Supercomputing (SC'02). Baltimore, Usa, Page 58. (2002).

A. Chervenak, E. Deelman, C. Kesselman, B. Allcock, I. Foster, V. Nefedova, J. Lee, A. Sim, A. Shoshani, B. Drach, D. Williams, and D. Middleton. High-performance remote access to climate simulation data: a challenge problem for data grid technologies. Parallel Computing, 29(10): pp. 1335–1356, (2003)

A. Chervenak, R. Schuler, C. Kesselman, and S. Koranda. Wide area data replication for scientific collaborations. International Journal of High Performance Computing and Networking, 5(3): pp. 124–134, (2008).

H. Chettaoui and F. Ben Charrada. A new decentralized periodic replication strategy for dynamic data grids. Scalable Computing: Practice and Experience, 15(1), pp. 101–119, (2014)

U. Cibej, B. Slivnik, B. Robic. The complexity of static data replication in data grids. J. of Parallel Computing, 31(8-9): pp. 900–912. (2005)

B. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H. Jacobsen, N. Puz, D. Weaver, R. Yerneni. Pnuts: Yahoo!'s Hosted Data Serving Platform. Int. Conf. of Very Large DataBases (VLDB), pp. 1277-1288, (2008).

Z. Cui, D. Zuo, Z. Zhang. Based on the correlation of the file dynamic replication strategy in multi-tier data grid. Int. Journal of Database Theory and Application, 8(1): pp.75–86, (2015)

C. Dabas, J. Aggarwal. Delayed replication algorithm with dynamic threshold for cloud datacenters. Applications of Computing, Automation & Wireless Systems in Electrical Engineering. pp. 625–637, Springer, (2019).

T. Da Silva, M. Nascimento, J. Macedo. Towards non-intrusive elastic query processing in the cloud. Int. Workshop on Cloud data management. ACM, Hawai, USA, pp. 9-16, (2012).

W. Dai, I. Ibrahim and M. Bassiouni. An Improved Replica Placement Policy for Hadoop Distributed File System Running on Cloud Platforms. IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud), New York, NY, pp. 270-275, (2017)

N. Devakirubai, A. Kannammal. Optimal Replica Placement in Graph Based Data Grids. The International Journal Of Engineering And Science (IJES), Vol. 2, Issue 3, pp. 95-103, (2013).

E. Deelman, C. Kesselman, G. Mehta, L. Meshkat, L. Pearlman, K. Blackburn, P. Ehrens, A. Lazzarini, R. Williams, S. Koranda. GriPhyN and LIGO, building a virtual data grid for gravitational wave scientists. 11 th IEEE International Symposium on High Performance Distributed Computing (HPDC), pp. 225–234, (2002).

D. J. DeWitt, J. Gray. Parallel database systems: The future of high performance database systems. Communication ACM 35(6), pp. 85–98, (1992),

A. Dogan. A study on performance of dynamic file replication algorithms for real-time file access in data grids. Future Generation Computer Systems, 25(8): pp. 829–839, (2009)

Z. Du, J. Hu, Y. Chen, Z. Cheng, and X. Wang. Optimized qos-aware replica placement heuristics and applications in astronomy data grid. Journal of Systems and Software, 84(7): pp. 1224–1232, (2011)

A. Dutt, H. Jain, and S. Kumar, "Providing Software as a Service: a design decision(s) model. Information Systems and E-Business Management. Vol. 16, N. 2, pp. 327–356, (2018)

E. B. Edwin, P. Umamaheswari, M.R. Thanka. An efficient and improved multi-objective optimized replication management with dynamic and cost aware strategies in cloud computing data center. Cluster Computing, Vol. 22, Supp. 5, pp. 11119-11128, (2019).

A. J. Elmore, S. Das, D. Agrawal, and A. El Abbadi. Zephyr: Live Migration in Shared Nothing Databases for Elastic Cloud Platforms. In ACM SIGMOD Int. Conf. on Management of Data, pp. 301-3012, (2011).

L. Ferreira, F. Coelho, J. Pereira. Self-tunable DBMS Replication with Reinforcement Learning. Remke A., Schiavoni V. (eds) Distributed Applications and Interoperable Systems. DAIS 2020. Lecture Notes in Computer Science, V. 12135. Springer, Cham, pp. 131-145, (2020)

I. Foster. The anatomy of the grid: Enabling scalable virtual organizations. International Journal of High Performance Computing Applications, 15(3): pp. 200–222, (2001)

I. Foster (editor), D. Berry, A, Djaoui, A. Grimshaw, B. Horn, H Kishimoto (editor), F. Maciel, A. Savva, F. Siebenlist, R. Subramania, J. Treadwell, J. Von Reich. The Open Grid Services Architecture, Version 1.0. Global Grid Forum. (2004)

I. Foster. The Grid: A New Infrastructure for 21st Century Science, Physics Today Journal. Vol. 55, N. 2, pp. 42-56, (2002).

I. Foster, Y. Zhao, I. Raicu, S. Lu. Cloud computing and grid computing 360-degree compared. In 2008 Grid Computing Environments Workshop, pages 1–10. IEEE, (2008)

X. Fu, X. Zhu, J. Han, R. C. Wang. QoS-aware replica placement for data intensive applications. Journal of China Universities of Posts and Telecommunications, 20(3): pp. 43–47. (2013)

H. Fu, Z. Li, C. Wu and X. Chu. Core-Selecting Auctions for Dynamically Allocating Heterogeneous VMs in Cloud Computing. IEEE 7th International Conference on Cloud Computing, Anchorage, AK, pp. 152-159., (2014).

F. Garcia-Carballeira, J. Carretero, A. Calderon, J.D. Garcia, L.M. Sanchez, A Global and Parallel File Systems for Grids, Future Generation Computer Systems 23 (1), pp. 116-122, (2007).

M. N. Garofalakis and Yannis E. Ioannidis. Multi-dimensional resource scheduling for parallel queries. *SIGMOD Rec.*, 25(2): pp. 365–376, (1996)

H. Ghanbari, B. Simmons, M. Litoiu, G. Iszlai. Exploring alternative approaches to implement an elasticity policy. IEEE Int. Conf. on Cloud Computing (CLOUD), pp. 716–723. (2011)

S. Ghemawat, H. Gobioff, S-T. Leung. The Google file system. ACM SIGOPS Operating Systems Review, (2003).

NK. Gill, S. Singh. A dynamic, cost-aware, optimized data replication strategy for heterogeneous cloud data centers. Future Gen. Comp. Syst., Elsevier, Vol. 65, pp. 10-32, (2016).

S. Goel, R. Buyya. Data Replication Strategies in Wide Area Distributed Systems," Enterprise Service Computing: From Concept to Deployment, pp. 211-241, (2006).

S. Gopinath and E. Sherly, A Comprehensive Survey on Data Replication Techniques in Cloud Storage Systems. International Journal of Applied Engineering Research ISSN 0973-4562 V. 13, N. 22, pp. 15926-15932 (2018a).

S. Gopinath and E. Sherly. A Dynamic Replica Factor Calculator for Weighted Dynamic Replication Management in Cloud Storage Systems. Procedia Computer Science. Vol. 132, pp. 1771-1780, (2018b).

K. Grace, R. Manimegalai. Dynamic replica placement and selection strategies in data grids A comprehensive survey. Journal of Parallel and Distributed Computing 74(2), pp. 2099- 2108 (2014).

G. Graefe, P. Alto, A. Nica, K. Stolze, T. Neumann, T. Eavis, I. Petrov, E. Pourabbas, A. Rubertiand, D. Fekete. Elasticity in Cloud Databases and Their Query Processing. International Journal of Data Warehousing and Mining (IJDWM), Vol. 9 Issue 2, pp. 1-20, (2013)

A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel. The cost of a cloud: Research problems in data center networks. Computer Communication Review, 39(1): pp. 68–73, (2009)

N. Grozev and R. Buyya. Performance modelling and simulation of three-tier applications in cloud and multi-cloud environments. Computer Journal, Vol. 58, Issue 1, (2015).

G. Guerrero-Contreras, C. Rodriguez-Dominguez, S. B. Diaz and J. Garrido. Dynamic Replication and Deployment of Services in Mobile Environments. New Contributions in Information Systems and Technologies, Publishing, pp. 855–864. (2015).

J. S. Gwertzman, M. Seltzer. The case for geographical push-caching. Proceedings of the Fifth Workshop on Hot Topics in Operating Systems (HotOS-V), IEEE, WA, USA, pp. 51.55. (1995)

A Hameurlain, P. Bazex, F. Morvan. Traitement parallèle dans les bases de données relationnelles, 312 pages, Editions CÉPADUÈS, (1996).

A Hameurlain, F. Morvan. Evolution of Query Optimization Methods. Transactions on Large-Scale Data and Knowledge-Centered Systems. Vol. 1, pp. 211-242 (2009)

A. Hameurlain, F. Morvan. Big data management in the cloud: evolution or crossroad? In: Kozielski, S., Mrozek, D., Kasprowski, P., Małysiak-Mrozek, B., Kostrzewa, D. (eds.) BDAS 15-16. CCIS, V. 613, . Springer, Cham, pp. 23–38 (2016)

A. Hameurlain, R. Mokadem. Guest editors. Preface in Special Issue on Elastical Data Management in Cloud Systems. Journal of Computer Systems Science and Engineering (IJCSSE). Vol. 32, Issue 4 (2017).

T. Hamrouni, S. Slimani, and F. Ben Charrada. A data mining correlated patterns-based periodic decentralized replication strategy for data grids. Journal of Systems and Software, 110: pp. 10–27, (2015)

T. Hamrouni, S. Slimani, and F. Ben Charrada. A critical survey of data grid replication strategies based on data mining techniques. Proc. Comput. Sci., 51, pp. 2779-2788, (2016)

R. Han, L. Guo, M. M. Ghanem, Y. Guo. Lightweight Resource Scaling for Cloud Applications. 12th IEEE/ACM Int. Symposium on Cluster, Cloud and Grid Computing (CCGrid), Ottawa, pp. 644-651, (2012)

O. Hassan, L. Ramaswamy, J. Miller, K. Rasheed, R. Canfield. Replication in overlay networks: a Multi-Objective Optimization Approach. Collaborative Comp.: Networking, Applications and Worksharing. Springer, pp. 512–28, (2009).

L. He, Z. Qian, F. Shang. A novel predicted replication strategy in cloud storage. J. of Supercomputing, V. 76, pp. 4838-4856, Springer, (2018).

S. He and X. Sun. A Cost-Effective Distribution-Aware Data Replication Scheme for Parallel I/O Systems. IEEE Transactions on Computers, Vol. 67, no. 10, pp. 1374-1387, (2018)

A. Horri, R. Sepahvand, Gh. Dastghaibyfard, A hierarchical Scheduling and Replication Strategy. Proc. Of Int. Journal of Computer Science and Network Security 8 (August), (2008).

H. I. Hsaio, M. S. Chen & P. S. Yu. On parallel execution of multiple pipelined hash-joins. Proc. of ACM-SIGMOD, Minneapolis, MN, May 24-27, pp. 185-196, (1994).

T.-Y. Hsu, A. Kshemkalyani, and M. Shen. Causal consistency algorithms for partially replicated and fully replicated systems. Future Generation Computer Systems, pp. 1118-1133, (2018).

K. Hwang, X. Bai, Y. Shi, M. Li, W. Chen, Y. Wu. Cloud Performance Modeling with Benchmark Evaluation of Elastic Scaling Strategies. Trans. on Par. and Distr. Systems, 27 (1), pp. 130-143, (2016).

C-W Huang, C-C Shih, W. Hu, B-T Lin, C-W Cheng. The improvement of auto-scaling mechanism for distributed database - A case study for MongoDB", Network Operations and Management Symposium (APNOMS) 2013 15th Asia-Pacific, pp. 1-3, (2013)

K. Huang, Li D, Sun Y (2014) CRMS: a centralized replication management scheme for cloud storage system, Proc 2014 IEEE/CIC Int Conf Commun China (ICCC); pp. 344–348, (2014)

D. S. Jayalakshmi and R. T. P. Ranjana. Dynamic data replication strategy in cloud environments. Int. Conf. on Advances in Computing and Communications (ICACC), pp. 102–105, doi: 10.1109/ICACC.2015.79. (2015)

Joint Enterprise Defense Infrastructure, https://www.lemondeinformatique.fr/actualites/lire-cloud-microsoft-gagne-le-mega-contrat-du-pentagone-aws-recale-76910.html (2019)

J. Janpet & Y-F Wen. Reliable and available data replication planning for cloud storage. Int. Conf. on Advanced Information Networking & Applications (AINA), IEEE, pp. 772–779, (2013)

N. Kabra and D. J. Dewitt. Efficient mid-query re-optimization of sub-optimal query execution plans. In ACM Sigmod Records, ACM, Vol. 27, pp. 106-117, (1998).

M. M. Kandi. Allocation de ressources élastique pour l'optimisation de requêtes. Thèse de doctorat. Université Paul Sabatier, Toulouse III, (2019)

M.R. Kaseb, M.H. Khafagy, I. A.Ali, E. M. Saad. An improved technique for increasing availability in Big Data replication. V. 91, February 2019, pp. 493-505, Future Generation Computer Sustems (2019)

I. Ketata, R. Mokadem, F. Morvan, A. Hameurlain. Efficient Semi-Automatic Maintenance of Mapping Between Ontologies in a Biomedical Environment. Int. Conference on Enterprise Information Systems (ICEIS 2010), Funchal, Madeira - Portugal, V. 1, (Eds.), INSTICC Press, (2010)

I. Ketata, R. Mokadem, F. Morvan. Biomedical Resource Discovery considering Semantic Heterogeneity in Data Grid Environments. Int. Conf. on Integrated Computing Technology (InTech 2011), Sao Carlos-Brazil, (2011a)

I. Ketata, R. Mokadem, F. Morvan. Resource Discovery Considering Semantic Properties in Data Grid Environments" (regular paper), dans : International Conference on Data Management in Grid and P2P Systems (GLOBE 2011), Toulouse, Springer, LNCS 6864, p. 61-72, (2011b)

I. Ketata, R. Mokadem, F. Morvan, Semantic Resource Discovery in Large Scale Environments. Journal of Digital Information Management 9 (3), (2011c)

I. Ketata, Méthode de découverte de sources de données tenant compte de la sémantique en environnement de grille de données. Doctorat de l'université de Toulouse, Univ. Pau Sabatier, Toulouse III, (2012)

B. Kemme, R. Jimenez-Peris, and M. Patino-Martinez, Database Replication. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, (2010).

L. M. Khanli, A. Isazadeh, T. N. Shishavan. PHFS: A dynamic replication method, to decrease access latency in the multi-tier data grid. Future Generation Computer Systems, 27(3): pp. 233–244, (2011)

S. Khatua, A. Ghosh, N. Mukherjee. Optimizing the utilization of virtual resources in cloud environment.Int. Conf. on Virtual Environments Human-Computer Interfaces & Measurement Systems (VECIMS), pp. 82–87. (2010)

A. Khelifa, T. Hamrouni, R. Mokadem, F. Ben Charrada. Triadic Concept Analysis-based Data Replication Strategy while satisfying Tenant Performance and provider Profit Gurantees. International Journal of High Performance Computing and Networking (IJHPCN), Inderscience. In press (2020)

A. Khelifa, T. Hamrouni, R. Mokadem, F. Ben Charrada. RCPP: Cloud Provider Profit-aware and Triadic Concept Analysis-based Data Replication Strategy for Tenant Performance Improvement. Invited Session on Service-oriented Data Science in Int. Conf. on Knowledge-Based and Intelligent Information and Engineering Systems (KES), 16-18 Sept., Verona, Italy, (2020).

S. Kirubakaran, S. Valarmathy, C. Kamalanathan. Data replication using modified D2RS in cloud computing for performance improvement. Journal of Theoretical and Applied Information Technology, Vol. 58, No. 2, pp. 460–470, (2013).

K. Kloudas, M. Mamede, N. Preguiça, R. Rodrigues. Pixida: Optimizing Data Parallel Jobs in Wide-Area Data Analytics. Proc. of VLDB'15, pp. 72- 83, (2015).

Y. Kouki, T. Ledoux. SLA-driven Capacity Planning for Cloud Applications. 4th Conf. on Cloud Computing Technology and Science (CloudCom). pp. 135-140, (2012).

Y. Kouki, Approche dirigée par les contrats de niveaux de service pour la gestion de l'élasticité du" nuage". PhD thesis, Nantes, Ecole des Mines. (2013)

K. A. Kumar, A. Quamar, A. Deshpande, S. Khuller. SWORD: workload-aware data placement andreplica selection for cloud data management systems. The VLDB Journal, 23(6): pp. 845-870, (2014)

P. Kunszt, E. Laure, H. Stockinger, K. Stockinger, File-based replica management, Future Generation Computer Systems 22 (1), pp. 115–123, (2005).

R. S. G. Lanzelotte, P. Valduriez, M. Zaït, and M. Ziane. Industrial-strength parallel query optimization: Issues and lessons. Information Systems, 19(4): pp. 311–330, (1994)

H. Lamehamedi, B. Szymanski, Z. Shentu and E. Deelman. Data replication strategies in grid environments. Int. Conf. on Algorithms and Architectures for Parallel Processing, pp. 378– 383. IEEE, (2002)

R. Lakshmi and A. S. Thanamani. Performance Evolution of Dynamic Replication ina Data Grid using DMDR Algorithm. International Journal of Engineering Research & Technology (IJERT), Vol. 5 Issue 10, (2019)

P. Lama, S. Wang, X. Zhou and D. Cheng. Performance Isolation of Data-Intensive Scale-out Applications in a Multi-tenant Cloud. IEEE International Parallel and Distributed Processing Symposium (IPDPS), Vancouver, BC, pp. 85-94, (2018)

A. Lazeb, R Mokadem, G Belalem. Towards a New Data Replication Management in Cloud Systems. Int. Journal of Strategic Information Technology and Applications (IJSITA) 10(2), pp. 1-20 (2019a)

A. Lazeb, R. Mokadem, G. Belalem. Economic Data Replication Management in the Cloud. 3rd National Study Day on Research on Computer Sciences (JERI), Saida, Algeria. CEUR Workshop Proc. 2351, (2019b).

M. C. Lee, F. Y. Leu and Y. Ping Chen. PFRF: An adaptive data replication algorithm based on star-topology data grids. Future Generation Computer Systems, 28(7), pp. 1045–1057, (2012).

J. Lee, J. Chung, D. Lee. Efficient data replication scheme based on hadoop distributed file system. International Journal of Software Engineering and Its Applications, 9(12): pp. 177–186, (2015)

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

W. Li, Y. Yang and D. Yuan. A Novel Cost-Effective Dynamic Data Replication Strategy for Reliability in Cloud Data Centres. – In 2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing, pp. 496–502, Sydney, Australia, IEEE. (2011)

R. Li, Y. Hu, and P. Lee. 2017. Enabling Efficient and Reliable Transition from Replication to Erasure Coding for Clustered File Systems. IEEE Trans. on Parallel and Distributed Systems, Vol. 28, Issue 9, pp. 2500—2513, (2017)

C. Li, Y. Wang, Y. Chen, Y. Lu. Energy-efficient fault-tolerant replica management policy with deadline and budget constraints in edge-cloud environment. J. of Network and Computer Applications, V. 143, pp. 152-166, (2019)

S. Limam, R. Mokadem, G. Belalem. Satisfying Availability and Performance Requirements while Ensuring Profit of Cloud Providers. Int Conf. on Embedded & Distributed Systems (EDiS), Oran, Algeria,(2017).

S. Limam. Vers une gestion efficace de réplication et de cohérence des données dans les Clouds Computing. Doctorat de l'université d'Oran 1 A. Ben Bella, (2018)

S. Limam, R. Mokadem, G. Belalem. Data replication strategy with satisfaction of availability, performance and tenant budget requirements. Journal of Cluster Computing. Springer USA, Vol. 22, N. 4, pp. 1199-1210, (2019)

H. Lin, J.H. Abawajy, R. Buyya, Economy-based data replication broker, in: Proceedings of the 2nd IEEE Int. Conf. on e-Science and Grid Computing, Amsterdam, Netherlands, (2006).

J-W. Lin, C-H. Chen, J-M. Chang. Qos-aware data replication for data-intensive applications in cloud computing systems. IEEE Trans Cloud Comput. Vol. 1, Issue 1, pp. 101-115, (2013)

Y. F. Lin, J.J. Wu, P. Liu, A list-based strategy for optimal replica placement in data grid systems, in: 37th International Conference on Parallel Processing, pp. 198–205, (2008).

W. Litwin, R. Mokadem, P. Rigaux, T. Schwarz. Fast nGram-Based String Search Over Data Encoded Using Algebraic Signatures. Int. Conf. on Very Large Data Bases (VLDB), University of Vienna. Austria, VLDB Endowment, pp. 207-218, (2007).

G. Liu, H. Shen, H. Chandler. Selective Data Replication for Online Social Networks with Distributed Datacenters. *I*EEE Transactions on Parallel and Distributed Systems. Vol. 27, N. 8, pp. 2377-2393, (2016).

J. Liu, H. Shen. A Low-Cost Multi-Failure Resilient Replication Scheme for High Data Availability in Cloud Storage. IEEE 23rd Int. Conf. on High Performance Computing (2016)

G. Liu and H. Shen. Minimum-cost cloud storage service across multiple cloud providers. IEEE/ACM Trans. Netw., vol. 25, no. 4, pp. 2498-2513, (2017).

J. Liu, H. Shen, H. S. Narman, Z. Lin, Z. Li. Popularity-aware Multi-failure Resilient and Cost-effective Replication for High Data Durability in Cloud Storage. Transactions on Parallel and Distributed Systems. (2018)

K. Liu, J. Peng, J. Wang, W. Liu, Z. Huang, J. Pan. Scalable and Adaptive Data Replica Placement for Geo-Distributed Cloud Storages. IEEE Trans. on Parallel and Distributed Systems, vol. 31, no. 7, pp. 1575-1587, (2020).

S-Q. Long, Y-L. Zhao and W. Chen. MORM: A Multi-objective strategy for cloud storage cluster. Journal of Systems Architecture, Vol. 60, Issue 2, pp. 234–244, (2014).

J. Ma, W. Liu, and T. Glatard. A classification of file placement and replication methods on grids. Future Generation Computer Systems, 29(6):, pp. 1395–1406. (2013)

K. Ma, B. Yang. Stream-based live data replication approach of in-memory cache. Concurr. Comput.: Practice and Experience, 29 (11), pp. 1-9, (2017)

J. P. Magalhaes, L. M. Silva. A Framework for Self-healing and Self-adaptation of Cloud-hosted Web-based Applications. IEEE International Conference on Cloud Computing Technology and Science, pp. 555-565, (2013).

N. Maltsev, E. Glass, D. Sulakhe, A. Rodriguez, M. H Syed, T. Bompada, Y. Zhang, M. D'Souza. Puma2–grid-based highthroughput analysis of genomes and metabolic pathways. Nucleic acids research, 34 (Database issue), pp. 369–372, (2006)

N. Mansouri and G. H. Dastghaibyfard. A dynamic replica management strategy in data grid. Journal of Network and Computer Applications, 35 (4): pp. 1297–1303, (2012)

N. Mansouri and G. Hosein Dastghaibyfard. Enhanced dynamic hierarchical replication and weighted scheduling strategy in data grid. Journal of Parallel and Distributed Computing, 73(4): pp. 534–543.. (2013)

N. Mansouri, M. Javidi. E. Mansouri. Combination of data replication and scheduling algorithm for improving data availability in Data Grids. J. of Network and Computer Applications. V. 36, Issue 2, pp. 711-722, (2013)

N. Mansouri. Adaptive data replication strategy in cloud computing for performance improvement. Frontiers of Computer Science, 10*(5)*, pp. 925–935, (2016)

N. Mansouri, M. K. Rafsanjani, M.M. Javidi. DPRS: A dynamic popularity aware replication strategy with parallel download scheme in cloud environments. Simul. Model. Theory, Vol 77, pp. 177-196, (2017)

N. Mansouri, N. M. Javidi. A Survey of Dynamic Replication Strategies For Improving Response Time In Data Grid Environment, AUT Journal of Modeling and Simulation, 49 (2), pp. 239-264, (2017).

N. Mansouri, N, M. M. Javidi. A hybrid data replication strategy with fuzzy-based deletion for heterogeneous cloud data centers. J Supercomput, Volume 74, Issue 10, pp. 5349–5372, (2018a)

N. Mansouri, M.M. Javidi. A new Prefetching-aware Data Replication to decrease access latency in cloud environment. The Journal of Systems & Software. Vol. 144, pp. 197-215 (2018b)

N. Mansouri, M.M. Javidi. An Efficient Data Replication Strategy in Large-Scale Data Grid Environments Based on Availability and Popularity. AUT Joural of Modeling and Simulation, 50(1), pp. 39-50, (2018c)

N. Mansouri, M.M. Javidi. A Review of Replica Replacement Techniques in Grid Computing and Cloud Computing. Journal of Algorithms and Computation, Vol 51 issue 2, pp. 134 – 151, (2019)

Y. Mansouri, A. N. Toosi, and R. Buyya. Cost optimization for dynamic replication and migration of data in cloud data centers. IEEE Transactions on Cloud Computing, V. 7, Issue 3, pp. 705–718. (2019)

Y. Mansouri, R. Buyya. Dynamic replication and migration of data objects with hot-spot and cold-spot statuses across storage data centers. J. of Parallel and Distributed Computing. Vol. 126, pp. 121-133, (2019)

S. S.Manvi and G. K. Shyam. Resource management for Infrastructure as a Service (IaaS) in cloud computing: A survey. Journal of Network and Computer Applications, Vol. 41, pp. 424-440, (2014)

B. Mao, S. Wu, H. Jiang. Exploiting workload characteristics and service diversity to improve the availability of cloud storage systems. IEEE Trans. Parall. Distrib. Syst. 27, 7, pp. 2010–2021, (2016)

R. Marcus, O. Papaemmanouil, S. Semenova, S. Garber. NashDB: An end-to-end economic method for elastic database fragmentation, replication, and provisioning. Int. Conf. on Management of Data, pp. 1253–1267, 2018

T. Mastelic, A. Oleksiak, H. Claussen, I. Brandic, J.-M.Pierson, and A. V. Vasilakos. Cloud computing: Survey on energy efficiency," ACM Comput. Surv., vol. 47, no. 2, Paper 33, December (2014)

Medical Cloud solution. https://www.medicalcloud.fr/ (Décembre, 2020)

P. Mell, T. Grance. The NIST definition of Cloud computing (2011).

H. Mengxing, Y. Xianglong, W. Sanpeng, and Z. Donghai. A strategy of dynamic replica creation in cloud storage. Int. Workshop on Cloud Computing and Information Security, pp. 389–392, Paris, Atlantis Press, (2013)

M. Miglierina, G. P. Gibilisco, D. Ardagna and E. Di Nitto. Model based control for multi-cloud applications. In ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems, pp. 37–43, (2013)

N. J. Milani, Navimipour. A Comprehensive Review of the Data Replication Techniques in the Cloud Environments: Major Trends & Future Directions. J. Netw. Comput. Appl. Vol. 64, pp. 229–238, (2016).

R. Mokadem, A. Hameurlain, F. Morvan. Equi Join Query Acceleration Using Algebraic Signatures. IADIS International Conference Applied Computing. Algarve, Portugal. (2008a).

R. Mokadem, A. Hameurlain, F. Morvan. Performance Improving of Semi-join Based Join Operation through Algebraic Signatures. IEEE Int. Symposium on Parallel and Distributed Processing with Applications (ISPA), pp. 431-438, (2008b)

R. Mokadem, A. Hameurlain, F. Morvan. Performance improvement of join queries through algebraic signatures. International Journal of Intelligent Information and Database Systems. IJIIDS 4(3): pp. 282-306, (2010a)

R. Mokadem, A. Hameurlain and A. Min Tjoa. Resource Discovery Service while Minimizing Maintenance Overhead in Hierarchical DHT Systems. Int. Conf. on Information Integration and Web-based Applications & Services (iiWAS), Paris, France. ACM, p. 628-636, (2010b)

R. Mokadem, A. Hameurlain and A. Min Tjoa. (2011). An Efficient Resource Discovery while Minimizing Maintenance Overhead in SDDS based Hierarchical DHT Systems. Int. Journal of Grid and Distributed Computing (IJGDC). Vol. 4, N. 3, pp. 1-24. (2011)

R. Mokadem, A. Hameurlain, A Min Tjoa. Resource Discovery Service while Minimizing Maintenance Overhead in Hierarchical DHT Systems (IJARAS), 3(2): pp. 1-17. (2012a)

R. Mokadem, F. Morvan, A. Hameurlain. SDDS based Hierarchical DHT Systems for an Efficient Resource Discovery in Data Grid Systems. Int. Workshop on REsource Discovery (RED) in conjunction with the ESWC Conference. Heraklion, Greece. CEUR proceeding, Vol 862, pp 52-66, (2012b).

R. Mokadem, F. Morvan, C. Ghedira, D. Benslimane. DSD: a DaaS Service Discovery Method in P2P Environment. East-European Conference on Advances in Databases and Information Systems (ADBIS'13), Genoa, Italy. Vol. 241, Springer, pp. 129-137. (2013).

R. Mokadem, A. Hameurlain. Data Replication Strategies with Performance Objective in Data Grid Systems: A Survey. Int. Journal of Grid and Utility Computing (IJGUC), Inderscience Publishers, Vol. 6 N. 1, pp. 30-46, (2015).

R. Mokadem, A. Hameurlain. A Data Replication Strategy with Tenant Performance and Provider Economic Profit Guarantees in Cloud Data Centers. Journal of Systems and Software (JSS), Elsevier, V. 159, https://doi.org/10.1016/j.jss.2019.110447, (2020).

R. Mokadem, J. Martinez-Gil, A. Hameurlain. J. Kueng. A Review on Data Replication Strategies in Cloud Systems. Soumis au Journal of Grid and Utility Computing (IJGUC), Inderscience. Accepted in July 2020. In Press. (2020).

H. J. Moon, H. Hacumus¸ Y. Chi, W.-P. Hsiung. Swat: A Lightweight Load Balancing Method for Multitenant Databases. Proc. of EDBT'13. USA, ACM, pp. 65–76, (2013).

T.S. Muthu, K. R. kumar. Hybrid predictive approach for replica replacement in data grid, In: International Conference on Computer Communication and Informatics, (2017).

J. Myint and T. T. Naing. Management of data replication for pc cluster based cloud storage system. *International* Journal on Cloud Computing: Services and Architecture (IJCCSA), 1(3): pp. 31– 41, (2011)

V. R. Narasayya, S. Das, M. Syamala, B. Chandramouli, and S. Chaudhuri. SQLVM: Performance isolation in multi-tenant relational database-as-a-service. Biennial Conf. Innovative Data Syst. Res., Online Proceedings [www.cidrdb.org]. (2013)

A. Naskos, A. Gounaris, I. Konstantinou. Elton: a cloud resource scaling-out manager for nosql databases. 34th IEEE Int. Conf. on Data Engineering (ICDE), IEEE, pp.1641–1644. (2018)

B. Nazir, f. Ishak, S. Shamshirband, A. T. Chronopoulos. The impact of the implementation cost of replication in data grid job scheduling. Math. Comput. Appl. 23(2), 28 (2018)

B. Nicolae. Leveraging naturally distributed data redundancy to reduce collective I/O replication overhead, IEEE Int. conf. on Parallel Distrib. Process. Symp, pp. 1023–1032. (2015)

C. Nicholson, D. G. Cameron, A. T. Doyle, A. Paul Millar, and Kurt Stockinger. Dynamic data replication in LCG 2008. Concurrency and Computation: Practice and Experience, 20(11): pp. 1259–1271, (2008)

D. T. Nukarapu, B. Tang, L. Wang, S. Lu. Data Replication in Data Intensive Scientific Applications with Performance Guarantee. Proc. Of IEEE Transactions on Parallel and Distributed Systems;22: pp. 1299–306, (2011).

K. Omer, GMT Abdalla. Dynamic Algorithms Replication Using Grid Computing. Int. Conference on Computer, Control, Electrical, and Electronics Engineering, pp. 1-6, (2018).

Oracle white paper. Parralel Execution with Oracle Database. February 20 (2019)

J. Ortiz, V. T. de Almeida, and M. Balazinska. Changing the face of database cloud services with personalized service level agreements Biennial Conf. Innovative Data Syst. Res., Jan. 4-7, Online Proceedings, [Online]. Available: www.cidrdb.org.

T. Ozsu and P. Valduriez, Principles of Distributed Database Systems, Fourth Edition. Springer, ISBN 978-3-030-26252-5, pp. 1-674, (2020).

E. Pacitti, C. Coulon, P. Valduriez, T. Ozsu. Preventive Replication in a Database Cluster. Distribited and Parallel Databases, Vol. 18, pp. 223–251, (2005)

E. Pacitti, P Valduriez, M Mattosso. Grid data management: Open Problems and News Issues. Int. Jour. Grid Computing, Springer, Vol. 5, pp. 273-281. (2007)

S. M. Park, J.H. Kim, Y.B. Ko, W.S. Yoon. Dynamic data grid replication strategy based on Internet hierarchy, in: Grid and Cooperative Computing, pp. 838–846, (2004).

J. M. Perez, F. García-Carballeira, J. Carretero, A. Calderón, and J. Fernández. Branch replication scheme: A new model for data replication in large scale data grids. Future Generation Computer Systems, 26(1), pp. 2–20 (2010)

V. Persico, A. Pescapé, A. Picariello , G. Sperlf. Benchmarking Big Data Architectures for Social Networks Data processing using Public Cloud Platforms. Future Generation Computer Systems, Volume 89, pp. 98-109, (2018)

A. Pokahr and L. Braubach. Elastic component-based applications in PaaS clouds. Concurency and computation: Practice and experience Vol. 28, Issue 4, pp. 1368-1384, (2016)

Q. Pu, C. Ananthanarayanan, P. Bodik, S. Kanadula, A. Akella, P. Bahl, I. Stoica. Low latency geo-Distributed data analytics. In SIGCOMM, pp. 421- 434, (2015).

D. Qin, R. Liu, J. Zhen, S. Yang, and E. Wang. Research on decentralized group replication strategy based on correlated patterns mining in data grids. The Machine Learning and Intelligent Communications Conference, (MLICOM), Shanghai, China, August 27-28, pp 293–302. Springer Int. Publishing, (2016)

Y. Qu and N. Xiong. RFH: A resilient, fault-tolerant and high-efficient replication algorithm for distributed cloud storage. Int. Conf. on Parallel Processing, pp. 520–529. IEEE, (2012)

K. Qu, L. M. Kaiyang and Y. Yang. A dynamic replica strategy based on Markov model for hadoop distributed file system (HDFS). Int. Conf. on Cloud Computing and Intelligence Systems (CCIS), IEEE, pp. 337-342, (2016)

E. Rahm, T. Stöhr. Analysis of Parallel Scan Processing in Parallel Shared Disk Database Systems. Proc. In the Int. EURO-PAR Parallel Processing Conf. LNCS 966, Springer, pp. 485-500, (1995)

R. M. Rahman, K. Barker, R. Alhajj. Replica placement in data grid: a multi-objective approach. In Grid and Cooperative Computing-GCC 2005, Springer Berlin Heidelberg, pp 645–656, (2005).

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

P. Ranganathan, K. Gharachorloo, S. Adve, and L. Barroso. Performance of database workloads on shared-memory systems with out-of-order processors. The 8th International Conference on Architectural Support for Programming Languages and Operating Systems, San Jose, California, (1998).

K. Ranganathan, I. Foster. Identifying Dynamic Replication Strategies for a High Performance Data Grid. Int. Workshop on Grid Computing, Berlin, Germany. Springer, pp. 75-86, (2001).

K. Ranganathan, A. Iamnitchi, I. Foster. Improving data availability through dynamic model-driven replication in large peer-to-peer communities. EEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02), pp. 376–381, (2002)

J. Rao, X. Bu, C. Xu, K. Wang. A Distributed Self-Learning Approach for Elastic Provisioning of Virtualized Cloud Resources. IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems, Singapore, pp. 45-54, (2011).

Q. Rasool, J. Li, S. Zhang. Replica placement in multi-tier data grid. In Eighth IEEE International Conference on Dependable, Autonomic and Secure Computing, pp. 103–108. (2009)

I. A. Ridhawi, N. Mostafa, W. Masri. Location-aware data replication in cloud computing systems. IEEE Int. Conf. on Wireless and Mobile Computing, Networking and Communications (WiMob),pp. 20–27, (2015)

N. Saadat, A. M. Rahmani. PDDRA: a new pre-fetching based dynamic data replication algorithm in data grids. Future Gener Comput Syst 28(4): pp. 666–681, (2012)

S. Sakr, L. Zhao, H. Wada, A. Liu. CloudDB AutoAdmin: Towards a truly elastic cloud-based data store. In IEEE International Conference on Web Services, pages 732–733. IEEE doi:10.1109/ICWS.2011.19, (2011).

S. Sakr and A. Liu. SLA-based and Consumer-Centric Dynamic Provisioning for Cloud Databases. Proc. of the IEEE Int. Conf. On Cloud Computing, pp. 360–367, (2012).

Salesforce solution. https://www.salesforce.com/fr/products/what-is-salesforce/ (décembre 2020)

K. Sashi and A. S. Thanamani. Dynamic replication in a data grid using a modified bhr region based algorithm. Future Generation Computer Systems, 27 (2): pp. 202–210, (2011)

M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur. Xoring elephants: Novel erasure codes for big data. VLDB Endowment, vol. 6, no. 5, pp. 325–336, (2013)

M. Séguéla, R. Mokadem, J. M. Pierson. Comparing energy-aware vs. cost-aware data replication strategy. In: International Green and Sustainable Computing Conference (IGSC). pp. 1–8. (2019a)

M. Séguéla, R. Mokadem, J-M. Pierson. Étude des Stratégies de réplication de données prenant en compte la consommation énergétique vs. le profit économique dans les systèmes Cloud. Conférence francophone d'informatique en parallélisme, architecture et système (COMPAS), Anglet (2019b)

M. Séguéla, R. Mokadem, J-M. Pierson. Stratégie de réplication de données statique pour prendre en compte la consommation énergétique des fournisseurs de Cloud. Conférence francophone d'informatique en parallélisme, architecture et système (COMPAS), Lyon (2020)

S. A. E. Selvi, R. Anbuselvi. RAAES : Reliability-Assured and Availability-Enhanced Storage for Cloud Environment. Int. Journal of Pure and Applied Mathematics, vol. 118, pp. 103–112. (2018)

D. Serrano, S. Bouchenak, Y. Kouki et al. SLA guarantees for cloud services. Future Generation Computer Systems. V. 54, pp. 233-246. (2016)

U. Sharma, P. Shenoy, S. Sahu, A. Shaikh. Kingfisher: Cost-aware elasticity in the cloud. Proceedings IEEE INFOCOM. DOI: 10.1109/INFCOM.2011.5935016, (2011)

A. Sharov, A. Shraer, A. Merchant and M. Stokely. Take me to your leader! Proceedings of the VLDB Endowment, 8(12), pp. 1490-1501, (2015)

A. Shehabi, S. Smith, D. Sartor, R. Brown, M. Herrlin, J. Koomey, E. Masanet, N. Horner, I. Azevedo, L. William.United States Data Center Energy Usage Report. OSTI.gov, (2016)

W. Shi, C. Wu, Z. Li. An Online Auction Mechanism for Dynamic Virtual Cluster Provisioning in Geo-Distributed Clouds. IEEE Transactions on Parallel and Distributed Systems, vol. 28, no. 3, pp. 677-688, (2017)

M. Shorfuzzaman, P. Graham and R. Eskicioglu. Adaptive popularity-driven replica placement in hierarchical data grids. The Journal of Supercomputing, 51(3): pp. 374–392, doi: 10.1007/s11227-009-0371-9. (2009)

K. Shvachko, H. Kuang, S. Radia, and R. Chansler, The Hadoop Distributed File System. Proc. of IEEE MSST'10, pp. 1-10. (2010).

J. Sidell, P. Aoki, S. Barr, A. Sah, C. Staelin, M. Stonebraker, A. Yu. Data Replication in Mariposa, Proc. of 17th Int. Conf. on Data Engineering, New Orleans, USA, pp. 485-494, (1996).

G. Silvestre, S. Monnet, R. Krishnaswamy, P. Sens. *AREN*: a popularity aware replication scheme for cloud storage. IEEE 18th International Conference on Parallel and Distributed Systems, pp. 189-196. (2012)

S. Sivasubramanian, M. Szymaniak, G. Pierre, M. Van Steen. Replication forWeb Hosting Systems. ACM Computer Survey, 36(3), pp. 1–44, (2004)

S. Slimani, T. Hamrouni, F. Ben Charrada. Service-oriented replication strategies for improving quality-of-service in cloud computing: a survey. Cluster Computing. (2020)

J. E. Smith, R. Nair. The architecture of Virtual Machines. Computers, 38(5), pp. 32-38, (2005)

S. Souravlas and A. Sifaleras. Trends in data replication strategies: A survey. The International Journal of Parallel, Emergent and Distributed Systems. Vol. 34, Issue 2, pp. 222-239, (2017a)

S. Souravlas and A. Sifaleras, Binary-tree based estimation of file requests for efficient data replication, IEEE Transactions on Parallel and Distributed Systems 28, pp. 1839-1852, (2017b)

F. R. C. Sousa, L.O. Moreira, J. S. Costa Filho, J. C. Machado. Predictive elastic replication for multi-tenant databases in the cloud. Concurrency and Computation: Practice and Experience, Vol. 30, Issue 16, pp. 1-15, (2018)

K. Stamou, V. Kantere, J. H. Morin. SLA data management criteria. In IEEE International Conference on Big Data. Silicon Valley, CA. IEEE, pp. 34–42, (2013)

V. Stantchev and C. Schröpfer. Negotiating and enforcing QoS and SLAs in grid and cloud computing. In Advances in grid and pervasive computing, pp. 25–35. (2009)

F. R. C. Sousa, J.C. Machado. Towards Elastic Multi-Tenant Database Replication with Quality of Service. Proc. of Int. Conf on Utility and Cloud Computing, (UCC), pp. 168-175. IEEE Computer Society, Washington, (2012)

E. Spaho, A. Barolli, F. Xhafa & L. Barolli. P2P Data Replication: Techniques and Applications. In Modeling and Proc. for Next Generation Big-Data Technologies. Springer, pp. 145–166, (2015)

M. Stonebraker, P. M. Aoki, W. Litwin, A. Pfeffer, A. Sah, J. Sidell, C. Staelin, and A. Yu. Mariposa: A Wide-Area Distributed Database System. VLDB Journal: Very Large Data Bases, 5(1): pp. 48–63, (1996)

M. Stonebraker, D. Abadi, D. Dewitt, S. Madden, E. Paulson, A. Pavlo, A. Rasin. Mapreduce and parallel dbmss: friends or foes? Communications of the ACM 53, N. 1, pp. 64–71. (2010)

D. Sun, Gui-Ran Chang, Shang Gao, Li-Zhong Jin, and Xing-Wei Wang. Modeling a dynamic data replication strategy to increase system availability in cloud computing environments. Journal of Computer Science and Technology, 27(2): pp. 256–272, (2012a)

D. Sun, G. Chang, C. Miao, and X. Wang. Building a high serviceability model by checkpointing and replication strategy in cloud computing environments. In 32nd IEEE International Conference on Distributed Computing Systems Workshops, pp. 578–587. (2012b)

S.Y. Sun, W. Yao, X. Yong Li. DARS: a dynamic adaptive replica strategy under high load Cloud-P2P. Fut. Gener. Comput. Syst. Vol. 78, pp. 31–40. (2018)

A. Swami and K. B. Schiefer. On the estimation of join result sizes. In Advances in Database Technology - EDBT ’94, pages 287–300. (1994)

K. Tabet, R. Mokadem, M. R. Laouar, and S. Eom. Data Replication in Cloud Systems: A survey. Int. Journal of Information Systems in the Service Sector, Vol. 8, Issue 3, pp. 7–33, (2017)

K. Tabet, R. Mokadem, M. R. Laouar. Towards a new data replication strategy in mongodb systems. Proc. of the 4th ACM Int. Conf. of Computing for Engineering and Sciences (ICCES'18), pp. 1-8, (2018).

K. Tabet, R. Mokadem and M. R. Laouar. A Data Replication Strategy for Document Oriented NoSQL Systems. Int. Journal of Grid and Utility Computing, pp. 53-62 .(2019)

K. Tabet. Réplication de données dans les systèmes Cloud : Application pour les projets territoriaux. Doctorat de l’université Larbi Tebessi, Tebessa (2018)

A. Takefusa, O. Tatebe, S. Matsuoka, and Y. Morita. Performance analysis of scheduling and replication algorithms on grid datafarm architecture for high-energy physics applications. Proc. of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC’03), pp. 34–43, (2003).

Z. Tan and S. Babu. Tempo: Robust and self-tuning resource management in multi-tenant parallel databases. Proc. VLDB Endow., 9(10) ; pp. 720-731, (2016)

X. Tang and J. Xu. QoS-aware replica placement for content distribution. IEEE Transactions on Parallel and Distributed Systems, V. 16, pp. 921–932, (2005)

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

O. Tatebe, Y. Morita, and S. Matsuoka. Grid datafarm architecture for petascale data intensive computing. In International Symposium on Cluster Computing and the Grid (CCGRID’02), pp. 102–110, (2002)

N. Tomov, E. Dempster, M.H. Williams, A. Burger, H. Taylor, P.J.B. King, and P. Broughton. Analytical response time estimation in parallel relational database systems. Parallel Computing, 30(2) , pp. 249–283, (2004).

U. Tos, R. Mokadem, A. Hameurlain, A. Tolga, S. Bora. Dynamic replication strategies in data grid systems: A survey. Journal of Supercomputing, Springer-Verlag, Heidelberg, Allemagne, V. 71 N. 11, pp. 4116-4140, (2015)

U. Tos, R. Mokadem, A. Hameurlain, T. Ayav, S. Bora. A Performance & Profit Oriented Data Replication Strategy for Cloud Systems, IEEE Conf. on Cloud and Big Data Computing, CBDCom, France, pp. 780–787, (2016)

U. Tos, R. Mokadem, A. Hameurlain, T. Ayav, S. Bora. Ensuring performance and provider profit through data replication in cloud systems. Cluster Computing, Vol. 21 Issue 3, pp. 1479-1492, (2018)

U. Tos, Réplication de données dans les systèmes de gestion de données à grande échelle. Doctorat de l’université de Toulouse, Univ. Paul Sabatier, Toulouse III (2017)

M. Tu, P. Li, L. Xiao, I. Yen, F.B. Bastani. Replica Placement Algorithms for Mobile Transaction Systems. Trans. on Knowledge and Data Engineering 18 (7), pp. 954-970, (2006)

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

P. Valduriez. Parallel database systems: Open problems and new issues. Distributed and Parallel Databases 1(2), pp. 137–165 (1993),

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

D. Vengerov, W. Way, S. Jose, and A. C. Menck. Join size estimation subject to filter conditions. VLDB Endowment, Vol. 8, N. 12, pp. 1530–1541, (2015)

S. Venugopal, R. Buyya, and K. Ramamohanarao. A Taxonomy of Data Grids for Distributed Data Sharing, Management, and Processing. ACM Computing Surveys, Vol. 38, No. 1, ACM Press, NY, USA, (2006)

D. Vijayakumar, K. G. Srinivasagan and R. Sabarimuthukumar. FIR3: A fuzzy inference based reliable replica replacement strategy for cloud Data Centers. International Conference on Computing and Network Communications (CoCoNet), Trivandrum, pp. 473-479, (2015)

A. Vulimiri, C. Curino, B. Godfrey, J. Padhye, G. Varghese. Global analytics in the face of bandwidth and regulatory constraints. In 12th USENIX symp. NSDI'15, pp. 323- 336, (2015)

Q. Wei, B. Veeravalli, B. Gong, L. Zeng, D. Feng. CDRM: A Cost-Effective Dynamic Replication Management Scheme for Cloud Storage Cluster. IEEE Int. Conf. on Cluster Computing (CLUSTER), pp. 188-196, (2010).

Z. Wu, M. Butkiewicz, D. Perkins, E. Katz-Basset, H. V. Madhyastha. Spanstore: Cost-effective Geo-Replicated Storage Spanning Multiple Cloud Services. In SOSP, pp. 292-308, (2013).

L. Wu and R. Buyya. Service Level Agreement (SLA) in Utility Computing Systems. Performance and dependability in service computing : concepts, techniques and research directions. 25 pages, (2010).

F. Xhafa, V. Kolici, A. Potlog, E. Spaho, L. Barolli, M. Takizawa. Data Replication in P2P Collaborative Systems. Proc. of the 7th Int. Conf. on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), pp. 49–57, (2012)

P. Xiong, Y. Chi, S. Zhu, H. J. Moon, C. Pu, H. Hacigumus. Intelligent Management of Virtualized Resources for Database Systems in Cloud Environment. Proc. of Int. Conf. of Data Engineering (ICDE), pp. 87–98. (2011)

L. Xu, P. Andrew, S. Sudipta, J. Li, G. R. Ganger. Reducing Replication Bandwidth for Distributed Document Databases. SoCC '15, Kohala Coast, HI, USA, pp. 222-235, (2015a)

Z. Xu, N. Deng, C. Stewart and X. Wang. CADRE: Carbon-Aware Data Replication for Geo-Diverse Services. 2015 IEEE International Conference on Autonomic Computing, Grenoble, pp. 177-186. (2015b)

S. Yin, A. Hameurlain, F. Morvan. Sla definition for multi-tenant dbms and its impact on query optimization. IEEE Transactions on Knowledge and Data Engineering, 30(11), pp. 2213–2226. (2018).

M. Zakarya and L. Gillam. Energy efficient computing, clusters, grids and clouds: A taxonomy and survey. Sustainable Computing: Informatics and Systems, Vol. 14, pp. 13–33, (2017)

L. Zeng, S. Xu, Y. Wang, K. Kent, D. Bremner, C. Xu. Toward cost-effective replica placements in cloud storage systems with qos-awareness. Software: Practice and Experience, V. 47, Issue 6, pp. 813-829, (2016)

Z. Zeng, B. Veeravalli. Optimal metadata replications and request balancing strategy on cloud data centers. J Parallel Distrib Comput; vol. 74, Issue 10: pp. 2934–2940, (2014).

H. Zhang, B. Lin, Z. Liu, and W. Guo. Data replication placement strategy based on bidding mode for cloud storage cluster. 11th Web Information System and Application Conference, pp. 207–212, (2014)

L. Zhang, Y. Deng, W. Zhu, J. Zhou, and F. Wang. Skewly replicating hot data to construct a power-efficient storage cluster. Journal of Network and Computer Applications. Vol. 50, pp. 168–179, (2015)

J. Zhang, B.S. Lee, X. Tang, C.K. Yeo, A model to predict the optimal performance of the hierarchical data grid, Future Generation Computer Systems 26 (1), pp. 1–11, (2010)

Y. Zhang, J Jiang, K Xu, X. Nie, M. j. Reed, H. Wang, G. Yao, , M. Zhang, K. Chen. BDS: a centralized near-optimal overlay network for inter-datacenter data replication. The 30th EuroSys Conference, pp. 1–14, (2018)

L. Zhao, S. Sakr, A. Liu. A framework for consumer-centric SLA management of cloud-hosted databases. IEEE Transcations on services computing. Vol. 8, Issue 4, pp. 534-549, (2015)

P. Zhao, J. Shang, J. Lin, B. Li, X. Sun. A Dynamic Convergent Replication Strategy Based on Distributed Hierarchical Systems. Int. Conf. on Network, Communication and Computing (ICNCC), pp. 7–13, (2017)

J. Zhou, J. Fan. JPR: Exploring Joint Partitioning and Replication for Traffic Minimization in Online Social Networks. IEEE 37th Int. Conf. on Distributed Computing Systems (ICDCS), pp. 1147-1156, (2017)