Gestion de donnée complexes pour la modélisation de niche écologique

Ndiouma Bame

Université Pierre et Marie Curie

Université Cheikh Anta Diop de Dakar

École doctorale Informatique, Télécommunications et Électronique (Paris)

Laboratoire d'Informatique de Paris 6 (LIP6) / Equipe de recherche : Bases de Données

École doctorale Mathématique-Informatique (Dakar)

Laboratoire d'Informatique de Dakar (LID) / Equipe de recherche : Bases de Données et Data Minining

Titre de la thèse :

Gestion de données complexes pour la modélisation de niche écologique

Présenté par Ndiouma BAME

Thèse de doctorat d'Informatique

Date de soutenance prévue : 19 juin 2015

Devant un jury composé de :

Rapporteurs :	Claudia	RONCANCIO	Professeur Université de Grenoble
	Pascal	MOLLI	Professeur Université de Nantes
Examinateurs :	Régine	VIGNES	Professeur UPMC
	Maude	MANOUVRIER	Maître de Conférences Université Paris-Dauphine
Directeur de thèse UPMC :	Bernd	AMANN	Professeur UPMC
Encadrant UPMC :	Hubert	NAACKE	Maître de Conférences UPMC
Directeur de thèse UCAD :	Samba	NDIAYE	Maître de Conférences UCAD (HDR)
Encadrant UCAD :	Idrissa	SARR	Maître Assistant UCAD

Remerciements

N'eussent été les conseils et soutiens des uns et des autres, ce travail n'aurait probablement jamais pu être mené à son terme.

Que toutes celles et tous ceux qui nous ont apportés leur précieux concours aient l'amabilité d'agréer nos sincères remerciements.

Les nôtres :

- A *M. Hubert NAACKE*, Maître de conférences à l'Université Paris VI, pour la confiance qu'il a porté sur ma personne en me proposant cette thèse. Je le remercie vivement pour sa disponibilité, son sens de l'écoute, son efficacité dans ses remarques et ses orientations, ses conseils et encouragements et pour son soutien permanent malgré la distance géographique qui nous séparait souvent.
- *A M. Bernd AMANN***,** Professeur à l'UPMC d'avoir accepté d'être mon directeur de thèse à l'UPMC. Je le remercie pour ses remarques pertinentes, ses corrections, ses conseils et sa sociabilité.
- A *M. Samba NDIAYE*, Maître de Conférences à l'UCAD, pour toute l'aide qu'il m'a apporté, ses commentaires, ses orientations, ses corrections et sa disponibilité malgré un calendrier chargé.
- A *M. Idrissa SARR,* Maître Assistant à l'UCAD, pour sa disponibilité, ses remarques, qui, malgré son calendrier occupé a fait preuve d'abnégation pour me recevoir dans son bureau ou répondre à mes mails.
- Aux *membres de jury* d'avoir accepté d'évaluer ce travail. Je commence par remercier sincèrement mes deux *rapporteurs* Madame *Claudia RONCANCIA* et Monsieur *Pascal MOLLI* qui m'ont honoré en acceptant de juger en profondeur ma thèse. Je les remercie pour leurs remarques et suggestions pertinentes et précises pour améliorer la qualité du manuscrit. Puis, je remercie sincèrement les examinatrices *Régine Vignes-LEBBE* et *Maude MANOUVRIER* pour l'intérêt qu'elles ont porté à évaluer mon travail.
- A *Stéphane GANCARSKI* pour le soutien précieux qu'il m'a apporté dans la rédaction et la clarté de ce manuscrit et pour sa sociabilité.
- A tous les membres de l'*équipe Base de Données* du LIP6 pour leurs ouvertures, leurs sociabilités et leurs esprits de partage, qui ont facilité mon intégration et mon épanouissement au sein de l'équipe. Mes sincères remerciements à *Anne DOUCET, à Amine BAAZIZI, à Nelly, à Camelia, à Yifan et Olivier CURE*.
- A tous les membres de l'équipe de GBIF France pour leur disponibilité depuis le début de cette thèse jusqu'à son aboutissement. Je les remercie pour les nombreux entretiens qu'ils m'ont accordés et pour toutes les informations qu'ils ont acceptées de mettre à ma disposition. Je remercie sincèrement *Régine VIGNE-LEBBE et Marie-Elise LECOQ*.
- A l'ensemble du corps professoral de la section informatique de l'UCAD. Je remercie sincèrement les membres du groupe de recherche Datamining et Base de données,

Modou GUEYE, Aliou BOLY, Bassirou NGOM, Ibrahima GUEYE, etc. Je remercie également le coordinateur de la section *Karim KONATE* et Monsieur *Djamal SECK* pour leurs conseils et leurs encouragements.

- Je remercie Mouhamadou Lamine BA pour ses conseils et sa disponibilité dans la préparation de mon exposé.
- Aux coordinateurs du Programme Doctoral International – Modélisation des Systèmes Complexes (PDI-MSC) qui a financé initialement cette thèse, ainsi qu'aux doctorants et anciens doctorants du PDI-MSC avec qui j'ai partagé des moments de bonheurs pendant les doctoriales de 2011, 2012 et 2013.
- Au service de coopération et d'action (SCAC) pour avoir financé mes séjours de recherche à Paris.

Mes remerciements vont à l'encontre de ma famille et mes parents qui m'ont toujours soutenu pour l'atteinte de cet objectif.

Mes remerciements

- A ma mère *Mariama BA* qui m'a toujours soutenu dans la discrétion pendant des moments difficiles. Je la remercie pour ses conseils, ses encouragements et sa patience. Je souhaite une très longue vie à cette maman que j'appelle ma sœur puisque nous avons partagé la même éducation de ma grand-mère *Aminata NIANG* (qu'elle repose en paix).
- A mon père *Sidy BAME* pour ses conseils, ses encouragements et sa bienveillance à l'égard de sa famille.
- A mon beau-père *Ibrahima DIAGNE* pour ses conseils et son encadrement depuis mon cursus primaire.
- A mon fils *Bébé Mouhamad* pour tout son temps que j'ai sacrifié pour la thèse. Je le remercie de ne pas être à ses côtés pendant sa naissance et de ne pas bien jouer avec lui à cause de la distance qui nous séparait souvent.
- A mon épouse *Sokhna Khady MBAYE* pour sa patience, sa tendresse et ses encouragements. Je remercie sincèrement cette grande dame qui est toujours derrière moi et qui me pousse toujours pour l'atteinte des objectifs visés ensemble.

Mes remerciements vont enfin à l'encontre de toute personne qui a contribué de près ou de loin à l'élaboration de ce travail.

Résumé. Le phénomène du Big Data est l'un des grands dés informatique de la décennie en cours. La quantité des données produites augmente constamment et rend leur traitement de plus en plus difficile à gérer. L'accès, l'interrogation et l'analyse de ces nouvelles masses de données sont essentiels pour élargir les connaissances du domaine y afférent et font parties des défis majeurs du Big Data. Ceci est particulièrement crucial dans des domaines tels que les média sociaux, la génomique, la climatologie, les réseaux énergétiques complexes, l'astronomie, l'écologie et la biodiversité. La gestion de ces masses de données nécessite l'utilisation de nouveaux systèmes de gestion de données disposant de méthodes d'accès performantes et permettant d'exprimer des requêtes avec des langages de haut niveau. Un aspect particulièrement difficile à prendre en compte est le comportement très versatile des utilisateurs. Cela génère des demandes très fluctuantes. Or les solutions actuelles n'ont pas été conçues pour s'adapter dynamiquement à ce type de situation. Cette thèse se déroule dans le contexte du GBIF, initiative visant à fédérer et partager les données de biodiversité produites par de nombreux fournisseurs à l'échelle mondiale. Le GBIF propose actuellement des services pour interroger les données et les visualiser. Toutefois, avec un nombre croissant de fournisseurs qui ajoutent de nouvelles données et d'utilisateurs qui expriment de nouveaux besoins d'interrogation, l'accès aux données du GBIF pose un double problème d'expressivité et d'efficacité difficile à résoudre. L'objectif principal de cette thèse, est de concevoir une solution qui offre un accès expressif et efficace à une très grande base de données, lorsque le nombre d'utilisateurs devient très grand. Dans cette thèse, nous proposons une solution générale qui tient compte des spécificités réelles du cas d'usage du GBIF. Les résultats sont les suivants : 1) La conception d'une architecture décentralisée qui mutualise un grand nombre de ressources de stockage et de calcul. L'architecture est conçue pour agréger simplement et dynamiquement de nouvelles ressources, coordonner le stockage et l'interrogation entre les ressources. Elle peut être couplée de manière non intrusive avec des sources de données existantes, tout en permettant de manipuler les données indépendamment des sources. 2) La définition d'une solution de répartition dynamique des données à la demande. Les données sont fragmentées en fonction des prédicats des requêtes portant ur les dimensions hiérarchiques des données. Le placement des fragments et leur éventuelle réplication s'adapte dynamiquement à la charge des requêtes tout en tenant compte de la taille des ressources de stockage disponibles, à l'aide d'une fonction de coût. 3) Un modèle pour l'exécution répartie des requêtes dans cette architecture. Une requête est traitée en parallèle lorsque les données sont réparties sur plusieurs machines. L'exécution est décentralisée de telle sorte que toute machine peut recevoir les requêtes de plusieurs utilisateurs et gérer leur exécution. 4) Une méthode d'optimisation de requête qui considère plusieurs paramètres susceptibles d'impacter le coût de la requête : la fluctuation et la disparité des charges, la disparité des capacités de calcul et des liens de communication, le schéma de répartition des données et les prédicats des requêtes. Ayant comme objectif de traiter la plupart des requêtes avec un temps de réponse borné, la solution

ajuste dynamiquement le schéma de placement et de réplication des données. 5) Une mise en œuvre de la solution proposée et sa validation expérimentale dans le cas réel de d'analyse de la biodiversité avec les données du GBIF et des requêtes d'experts du domaine. Les résultats obtenus, en utilisant un cluster de 200 cœurs, montrent la faisabilité de la solution décentralisée non-intrusive pour le traitement réparti de requêtes complexes. Ils montrent le bénéfice d'adapter dynamiquement le schéma de placement des fragments en fonction de leur popularité et de la charge des sites. Les performances de notre solution sont satisfaisantes : notre approche s'avère efficace pour garantir un temps de réponse borné et un débit acceptable. Les résultats de l'évaluation du passage à l'échelle sont prometteurs.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

1.1 Motivations

1.1.1 Contexte et problématiques

Le phénomène du *Big Data* est de plus en plus perçu comme l'un des grands défis informatiques de la décennie en cours. De nombreux domaines font face à un déluge de données sans précédent. La quantité des données produites augmente constamment et rend leur traitement de plus en plus difficile avec les outils actuels. L'accès, l'interrogation et l'analyse de ces nouvelles masses de données sont essentiels pour élargir les connaissances du domaine y afférentes et font partie des défis majeurs du Big Data [Cen]. Ceci est particulièrement crucial dans des domaines tels que les média sociaux, la génomique, la climatologie, les réseaux énergétiques complexes, l'astronomie, l'écologie et la biodiversité. Par exemple Facebook enregistre plus 800 millions d'utilisateurs actifs en moyenne par jour [Fac] et ajoute quotidiennement plus de 2.7 millions de *Like* et plus 300 millions de photos [Tec]. La base de données astrométrique européenne est complétée avec plus de 6 millions de données de nouvelles observations par jour [Pro].

Dans cette thèse, nous nous intéressons en particulier au domaine de la biodiversité. Le système d'information, GBIF (pour Global Biodiversity Information Facility), une initiative présentée en détail à la section 3.1, fédère et partage les données de biodiversité produites par de nombreux fournisseurs à l'échelle mondiale. Les données collectées par le GBIF décrivent principalement des occurrences de specimens vivants. Les données sont structurées : une occurrence est un nuplet dont les attributs décrivent la classification taxinomique du specimen, son lieu et sa date d'observation, l'observateur, le fournisseur, etc. En termes de taille, la base de données contient plus de 500 millions d'occurrences issues de plus de 14000 collections et 650 fournisseurs [Sec].

Ces données servent de matière première à une large communauté des chercheurs qui étudient plusieurs aspects de la biodiversité comme, par exemple, l'impact du réchauffement climatique sur certaines espèces.

Le GBIF propose actuellement des services pour interroger ces données. Un utilisateur peut rechercher des occurrences en fonction de certains critères simples (nom scientifique d'un specimen, date et lieu d'observation). Il peut télécharger le résultat d'une requête afin de poursuivre son analyse avec ses propres outils. Un écologue qui étudie les abeilles invoquera le GBIF pour télécharger l'ensemble des occurrences d'abeilles. Puis, il devra préparer les données téléchargées pour les rendre compatibles avec ses outils qui calculeront la densité d'abeilles dans chaque zone étudiée. En plus des services d'interrogation et de téléchargement de données, le GBIF permet à l'utilisateur de visualiser sur une carte la position géographique des occurrences relatives à un pays ou à un fournisseur. Cette visualisation est figée et pré-calculée et un utilisateur ne peut ni filtrer les occurrences visualisées, ni choisir l'échelle de la visualisation. En résumé, les services offerts par le GBIF sont une première réponse au besoin de partager des données dans de nombreux travaux de recherche en biodiversité ([GBI]).

Avec un nombre croissant de fournisseurs qui ajoutent de nouvelles données et d'utilisateurs qui expriment de nouveaux besoins d'interrogation, l'accès aux données du GBIF commence à poser des problèmes d'*expressivité* et d'*efficacité* difficiles à résoudre.

Le manque d'expressivité empêche l'utilisateur d'exprimer des opérations pour filtrer, combiner, agréger les données avant de les télécharger. Par exemple, le scientifique ne peut pas demander à télécharger les fleurs localisées dans une région de France ayant également une forte densité d'abeilles. Il devra télécharger toutes les fleurs et toutes les abeilles de France, afin de les combiner pour ne garder finalement qu'une faible portion des données. Dans ce cas, la majeure partie des données a été transférée inutilement. Le manque d'efficacité est aussi mis en évidence quand des données sont modifiées. Lorsque le GBIF complète sa base avec une nouvelle collection de fleurs, chaque utilisateur concerné ré-exécute sa chaîne de traitement en commençant par télécharger les nouvelles données. Ceci génère une diffusion massive des nouvelles données qui pourrait être évitée si les utilisateurs partageaient un outil commun pour évaluer leurs requêtes. Le problème d'efficacité est d'autant plus crucial que le service de requêtes centralisé du GBIF pose plusieurs limites pour le *passage à l'échelle* face à la croissance des données et des demandes des usagers. En effet, la quantité de données partagées croit très rapidement. Le nombre d'enregistrements accessibles dans le portail est passé de 160 millions en décembre 2008 à 400 millions en 2013 pour atteindre plus 520 millions en mars 2015 [Sec]. La croissance du nombre d'enregistrement s'accentue à l'heure actuelle pour atteindre prochainement le milliard d'enregistrements. De plus, l'augmentation de l'expressivité génère de nouveaux traitements plus intensifs, alors que l'accès centralisé ne garantit pas le passage à l'échelle des traitements et données.

1.1.2 Objectifs

Pour résoudre les problèmes d'expressivité et d'efficacité, deux exigences concernant l'interrogation des données et la disponibilité du service doivent principalement être considérées :

1. Interrogation des données. Un service de requêtes doit permettre à l'utilisateur de déterminer précisément les données qui l'intéressent. Il doit disposer de méthodes d'accès permettant d'exprimer des requêtes dans un langage de haut niveau, comme les langages de requêtes déclaratifs offerts par les SGBD tels que SQL, XQuery ou Sparql.
2. Disponibilité du service. Un service de requêtes doit continuer à fonctionner convenablement même lorsqu'il est fortement sollicité par de nombreux utilisateurs. Ceci implique la possibilité d'exploiter un ensemble de ressources potentiellement immense, pour répartir les données et les traitements afin de répondre aux requêtes des utilisateurs en un temps limité.

1.1.3 Contraintes

Pour satisfaire les exigences d'interrogation et de disponibilité du service de requêtes, la solution doit tenir compte de plusieurs contraintes techniques liées à l'architecture et à l'usage.

- L'architecture de la solution doit :
  - pouvoir fonctionner avec des *ressources limitées et éloignées*, qui, cumulées, fournissent une grande puissance de calcul et de stockage avec un coût minimal en utilisant les ressources existantes des antennes nationales du GBIF. En particulier, l'architecture doit apporter des solutions d'accès au GBIF pour les chercheurs des pays du Sud qui ne disposent pas de ressources informatiques dédiées. Dans ce cas, il convient d'exploiter les ressources informatiques existant dans les antennes du GBIF (par exemple les machines de l'IFAN à Dakar) qui ne sont pas pleinement utilisées. Ces machines ne servent actuellement qu'à transmettre épisodiquement de nouvelles données au portail du GBIF. Une utilisation plus intensive de ces ressources est encouragée par le GBIF lui-même. Cette thèse s'inscrit dans cette démarche en proposant de fédérer les multiples ressources des antennes du GBIF pour offrir un service d'accès avancé à un plus grand nombre de chercheurs.
  - être *non-intrusive,* c'est à dire ne pas nécessiter la modification du portail GBIF qui coordonne l'intégration des nouvelles données.
  - être *autonome* pour traiter efficacement les requêtes des utilisateurs.
- Pour ce qui concerne les contraintes d'usage, la solution doit :
  - *disposer d'un langage d'interrogation de haut niveau interactif et expressif* pour les utilisateurs.

- *borner les temps de réponse* des requêtes pour satisfaire l'exigence de disponibilité du système et traiter les requêtes interactives des utilisateurs dans des délais impartis.
- *s'adapter dynamiquement au fil du temps* aux comportements versatiles des utilisateurs qui génèrent des demandes disparates et fluctuantes : la charge est variable en nombre de requêtes, certaines données sont plus populaires (fréquemment demandées) que d'autres, la popularité est elle-même fluctuante. Les données interrogées par un utilisateur changent au cours du temps en fonction des études et la fréquence d'accès aux données pour l'utilisateur varie en fonction de l'avancement de ses travaux. Par exemple, les requêtes concernant un agent (taxon) transmetteur d'une maladie augmente rapidement pendant la période de l'épidémie de la maladie. Ainsi lors de l'épidémie d'Ebola de 2014, les interrogations sur les chauve-souris (Chiroptera) ont augmenté d'une manière significative. Ce type d'événement engendre des fortes demandes sur les occurrences du taxon, qui diminuent progressivement avec la réduction de l'ampleur de l'épidémie.

1.1.4 Inadéquation des solutions actuelles

Pour interroger des données avec un langage de requêtes de haut niveau, une méthode habituelle consiste à importer toutes les données dans un système de gestion de base de données relationnelles (SGBDR) centralisé capable d'évaluer des requêtes SQL. Nous pouvons considérer un SGBDR comme une unité élémentaire de traitement de requêtes avec une capacité donnée pour la quantité de données qu'il peut stocker et pour le nombre de requêtes qu'il peut traiter dans une période de temps. De ce fait, un SGBDR ne passe pas à l'échelle.

Afin de garantir la disponibilité du service de requêtes, des approches existent pour répartir les données et les traitements entre plusieurs machines.

- Une première solution est d'opter pour un SGBDR parallèle composé de plusieurs unités SGBDR tel que MySQL Cluster, Parallel Data Warehouse, Greenplum, IBM DB2 Parallel Edition, Netezza, Oracle RAC Cluster, Teradata. Les données sont réparties entre les SGBDR de manière fixée initialement lors de la création du schéma de la base de données. Par exemple, on peut choisir une répartition taxonomique (ou géographique) des données de biodiversité pour partitionner les données par famille d'espèces (ou par pays) et placer les différents fragments à travers des sites dédiés avant l'utilisation du système. Les SGBDR parallèles sont bien adaptés si les requêtes peuvent être séparées en des groupes disjoints accédant à des parties différentes des données de façon régulière. Dans notre contexte le besoin de disponibilité n'est pas uniforme entre les fragments de données.Les SGBDR parallèles ne sont pas capables de repartitionner automatiquement les données en fonction des requêtes au fil du temps (Teradata, Greenplum, MySQL Cluster). Les SGBDR parallèles tel que SCOPE, IBM DB2 Parallel Edition, qui proposent un repartitionnement de données, se basent sur

les statistiques des tailles des données sur des intervalles de temps. Ceci est utile lorsque les statistiques correspondent aux comportements des applications dans les intervalles de temps qui suivent. Or, ceci n'est pas garanti avec le comportement très versatile des utilisateurs du GBIF. Une autre contrainte que les SGBDR parallèles imposent, est qu'ils fonctionnent avec des ressources coûteuses et localisées au même endroit.

– La deuxième solution est d'opter pour un système NoSQL tel que Hadoop, Hive, Spark SQL ou Cassandra, conçu pour passer à l'échelle. Ces systèmes ont l'avantage d'être extensibles et tolérant aux pannes : ils permettent d'ajouter facilement plusieurs machines. Ils peuvent ainsi stocker une quantité croissante de données et traiter un nombre croissant de requêtes. Les données sont réparties initialement entre les machines sans possibilité de repartitionner automatiquement les données en cours d'utilisation. Par exemple, Cassandra ne peut pas repartitionner les données sans recharger la base entière. Spark SQL qui propose un repartitionnement des données en permettant le déplacement des résultats intermédiaires pour minimiser le coût des jointures, utilise les statistiques des tailles données pour décider leur emplacement. Même si cette approche permet d'optimiser le coût local d'une jointure, elle ne considère pas pour autant la charge et les performances du site qui doit l'exécuter. Et, de ce fait, elle ne garantit pas un temps de réponse borné. Par ailleurs, si la plupart de ces systèmes sont conçus pour des applications de types *batch* et ne permettant pas à un utilisateur de poser des requêtes de manière interactive, les autres ne sont pas encore optimisés pour traiter efficacement des requêtes SQL complexes. Par exemple, la requête permettant de sélectionner les zones avec des fleurs et des abeilles dure 10 fois plus longtemps avec Spark SQL qu'avec un SGBDR en mémoire (voir les expériences comparatives dans la section 7.6).
– La troisième solution est d'opter pour une solution P2P composée de plusieurs pairs, tel que eMule, KaZaa, Gnutella, Chord ou PinS [VRL04], pour partager les données et passer à l'échelle. Ces systèmes sont caractérisés par de très grandes capacités de calcul et de stockage, ainsi que l'autonomie et le comportement dynamique des pairs. En effet, ils cumulent les capacités limitées des pairs qui peuvent rejoindre ou quitter le système à tout moment. Les schémas de données des pairs sont généralement hétérogènes et la disponibilité permanente des données hébergées sur un pair n'est pas garantie. De plus, le contenu d'un pair n'est pas adapté dynamiquement en fonction du comportement des autres utilisateurs. Une autre contrainte est que ces systèmes sont principalement destinés aux applications de partages de fichiers et sont limités pour interroger les données avec des requêtes complexes. Avec leur modèle de résolution de requêtes basé sur l'inondation, ils ne peuvent pas borner le temps de réponse pour une requête.

L'inconvénient principal de ces trois approches est que la répartition des données ne s'adapte pas au fil du temps en fonction de l'évolution des requêtes. Un inconvénient secondaire commun aux SGBDR parallèles et aux systèmes NoSQL, est que les ressources doivent être localisées

au même endroit (*i.e. un cluster de machines*), ce qui est incompatible avec notre contexte car nous ne voulons pas imposer une contrainte sur la localisation des ressources.

Une autre approche possible consiste à ne pas répartir les données mais à répliquer totalement la base du GBIF autant de fois que nécessaire pour des groupes distincts d'utilisateurs, de telle sorte que chaque SGBDR reçoive peu de demandes et les traite rapidement. Chaque SGBDR doit contenir toutes les données du GBIF car une requête peut interroger tout le contenu de la base. L'inconvénient de cette approche est de multiplier d'autant la quantité de données à diffuser pour maintenir à jour les répliques de la base du GBIF. De nouvelles données sont sans cesse diffusées et beaucoup de données seraient diffusées inutilement quand elles ne sont pas interrogées par la suite. De plus, cette approche ne s'applique pas si la taille des données dépasse la capacité d'un SGBDR, ce qui est de plus en plus fréquent.

Un premier bilan est que les approches existantes qui mettent en commun les ressources de plusieurs machines (ou plusieurs unité de traitement de requêtes en général) n'exploitent pas assez les caractéristiques des requêtes et les disponibilités disparates et fluctuantes des ressources pour réorganiser dynamiquement les données. Dans le contexte du GBIF, l'utilisateur n'accède pas toujours aux mêmes données : les critères de recherche changent en fonction des études. Les solutions actuelles ne sont pas adaptées à ce contexte dynamique surtout lorsque le nombre d'utilisateurs devient élevé.

Dans cette thèse, nous étudions l'interrogation efficace des données de biodiversité du GBIF dans un environnement réparti composé de plusieurs ressources limitées, existantes, éloignées et hétérogènes. Les principaux axes de cette étude sont énumérés dans la section suivante.

1.2 Questions de recherches

Nous faisons face à un problème de disponibilité des données sollicitées par les requêtes et de choix des portions de données à stocker dans les différentes machines au fil du temps. Ce problème est encore plus difficile lorsque les machines ont des petites capacités de stockage et de traitement ou que le coût de transfert des données est élevé. Par exemple, doit-on supprimer les données qu'un utilisateur a fini d'interroger, ce qui nécessite de les re-transférer (tâche coûteuse) la prochaine fois qu'il en aura besoin ? Est-il intéressant de répliquer une donnée très fréquemment accédée au détriment d'autres données ?

L'objectif de notre thèse est de concevoir une solution de partage et d'interrogation de données de biodiversité à large échelle qui surmonte les limites du GBIF. Pour parvenir à cette solution en tenant compte des contraintes architecturales et d'usage, nous tentons de répondre aux questions suivantes :

– Quelle architecture permet de traiter des requêtes plus expressives tout en garantissant le passage à l'échelle des applications et des données avec des ressources limitées éloignées ?
– Quelle stratégie de répartition de données est adaptée au contexte d'analyse des données de biodiversité pour garantir la disponibilité des données sollicitées ?
– Quelle méthode concevoir pour traiter une requête lorsque les données peuvent être réorganisées dynamiquement et pour bénéficier du traitement réparti d'une requête ?
– Quelle stratégie d'optimisation de requêtes peut garantir un temps de réponse borné pour la plupart des requêtes ?

1.3 Validation de la problématique auprès du GBIF

Au cours de la thèse, nous avons effectué des entretiens avec l'équipe de GBIF France pour déterminer les problèmes liés au fonctionnement du portail du GBIF pour l'interrogation des données de biodiversité du GBIF. Ces rencontres nous ont permis de comprendre le fonctionnement du portail GBIF (de l'intégration des données des fournisseurs à la mise à disposition des données au grand public) et de la nécessité d'optimisation de l'interrogation des requêtes des utilisateurs. En outre, grâce à la collaboration avec le GBIF France, nous avons eu des contacts avec des usagers des données du GBIF qui nous ont expliqué la chaine de traitements effectuée pour parvenir à leurs analyses avec les données du GBIF. Ceci nous a permis d'identifier les problèmes de l'usage limité des données, de l'extraction des occurrences pour l'analyse hors GBIF et la nécessité d'optimiser le traitement des requêtes pour un usage interactif du portail.

Ces rencontres étaient répétées pour actualiser les problématiques de recherche et nous ont permis de baser notre approche de répartition des données sur la structure multidimensionnelle des données de biodiversité.

1.4 Contributions

Les principales contributions de cette thèse sont :

– **Architecture décentralisée non-intrusive**
  Nous avons conçu une architecture décentralisée qui mutualise un grand nombre de ressources de stockage et de traitement de données existantes. Partant du constat que les ressources disponibles sont très hétérogènes (disparité de leur capacité de stockage, traitement et communication) et réparties à large échelle. L'architecture est conçue pour agréger simplement et dynamiquement des nouvelles ressources, coordonner le stockage et l'interrogation des données entre les ressources. L'architecture proposée peut être couplée de manière non intrusive avec des sources de données existantes, tout en permettant de manipuler les données

indépendamment des sources. Elle peut aussi agréger des ressources proches (cluster) ou fortement éloignées (multi-cloud).

- **Stratégie de répartition dynamique des données à la demande**
  Nous avons défini une solution de répartition dynamique des données à la demande. Les données sont fragmentées en fonction des requêtes. Les prédicats pour la fragmentation portent sur les dimensions (attributs) hiérarchiques des données. Si nécessaire les fragments peuvent être répliqués. Le placement des fragments et leur éventuelle réplication s'adapte dynamiquement en fonction des requêtes tout en tenant compte de la taille maximale des ressources de stockage disponibles, à l'aide d'une fonction de coût.
- **Traitement réparti de requêtes d'analyse**
  Nous proposons un modèle de traitement réparti de requêtes d'analyse. Le modèle de requêtes généralise le calcul de la cooccurrence de plusieurs taxons. L'exécution est décentralisée, gère simultanément les requêtes de plusieurs utilisateurs et coordonne la réplication dynamique des données.
- **Optimisation de requêtes basée sur un modèle de coût dynamique**
  Avec une contrainte de temps de réponse bornée, nous mettons en œuvre une approche d'optimisation de requêtes qui considère plusieurs paramètres susceptibles d'impacter le coût de la requête : fluctuation et disparité des charges, disparité des capacités de calcul et des liens de communication, localisation des données impliquées, schéma de fragmentation des données et prédicats des requêtes. Ayant comme objectif de ne pas surcharger un site, notre approche d'optimisation contrôle dynamiquement le placement des données. Les fragments de la requête qui se trouvent sur des sites qui seraient surchargés par la requête, sont répliqués vers des sites disponibles. Pour bénéficier du parallélisme du traitement réparti d'une requête, elle pousse au plus possible les opérations vers les données impliquées. Toutefois, elle favorise le traitement local de la requête, lorsque cela est possible pour garantir un temps de réponse borné.
- **Validation expérimentale et résultats**
  Nous avons implémenté notre solution et l'avons validée expérimentalement avec des données réelles de biodiversité du GBIF sur un cluster d'une dizaine de machines (200 cœurs). Les résultats obtenus montrent la faisabilité de la solution décentralisée non-intrusive pour le traitement réparti de requêtes complexes. Ils montrent que la solution adapte dynamiquement le schéma de placement de chaque fragment en fonction de sa popularité et des charges des sites qui contiennent des répliques de celui-ci. Les performances de notre solution sont satisfaisantes et montrent l'efficacité de notre approche d'optimisation pour garantir des temps de réponse borné avec un débit acceptable. Les résultats de l'évaluation du passage à l'échelle sont prometteurs. Ils montrent que les performances sont proportionnelles aux nombre de sites qui composent le système.
  Les résultats obtenus au cours de cette thèse ont fait l'objet de plusieurs publications : [BNSN12], [BNSN13], [BNSN14a], [BNSN14b], [SNB+14] .

Ces articles montrent la faisabilité de notre architecture décentralisée non-intrusive pour la répartition dynamique des données à la demande. Ils présentent des algorithmes de calcul de plan d'exécution des requêtes avec des validations par simulation. Entre temps nous avons amélioré ces travaux notamment ceux sur l'optimisation de requêtes que nous avons soumis pour publication.

1.5 Organisation du manuscrit

Cette thèse est organisée en huit chapitres.

- Le chapitre 1 présente l'introduction de notre thèse : les motivations, le contexte, les problématiques et les contributions.
- Le chapitre 2 présente l'état de l'art des travaux liés à notre thèse, concernant la répartition de données et l'optimisation de requêtes à large échelle et les solutions d'exploitation des données de biodiversité.
- Le chapitre 3 présente le GBIF et l'architecture décentralisée que nous proposons pour l'interrogation des données de biodiversité.
- Le chapitre 4 présente notre stratégie de répartition dynamique à la demande des données.
- Le chapitre 5 décrit le traitement réparti des requêtes d'analyse de données de biodiversité.
- Le chapitre 6 présente notre approche d'optimisation de requêtes basée sur un modèle de coût dynamique.
- Le chapitre 7 détaille la validation expérimentale de notre solution. La validation concerne la faisabilité de la solution non-intrusive et l'évaluation des performances et du passage à l'échelle.
- Le chapitre 8 montre la conclusion et les perspectives de cette thèse.

Chapitre 2

État de l'art

La gestion des données dans un environnement réparti à large échelle nécessite des méthodes d'accès aux données performantes et conçues pour passer à l'échelle. Dans cette thèse, nous considérons deux aspects importants de la gestion de données à large échelle que sont : la répartition des données et à l'optimisation des requêtes. Si la répartition définit l'organisation des données à travers les sites, l'optimisation quant à elle, décide de la manière dont ces données sont traitées pour satisfaire les requêtes qui les interrogent.
Dans la suite de ce chapitre, nous présentons, d'abord l'état de l'art de la répartition de données, ensuite, les travaux liés à l'optimisation de requêtes à large échelle et enfin, les solutions visant à améliorer l'exploitation des données de biodiversité.

2.1 Répartition de données à large échelle

Le fonctionnement d'une architecture répartie nécessite une bonne stratégie de répartition des données à travers les nœuds du système. La répartition des données joue un rôle essentiel sur les performances d'un système réparti de traitement de requêtes.

Dans cette section, nous allons, d'abord présenter quelques concepts inhérents à la répartition de données dans le contexte d'un environnement réparti. Ensuite, nous établirons l'état de l'art de la répartition dynamique de données. Enfin, nous positionnerons notre approche de répartition dynamique de données par rapport aux travaux menés dans le sens de la répartition dynamique de données à large échelle.

2.1.1 Concepts

Le processus de répartition des données dans un environnement réparti s'effectue en plusieurs étapes. La fragmentation est la première phase du processus de répartition de données .Elle consiste à découper la base de données globale en un ensemble de portions plus petites appelées fragments selon des critères bien définis. Ensuite, la stratégie de placement des fragments permet de déterminer le site qui doit héberger chaque fragment pour les traitements sur celui-ci. Afin de garantir la disponibilité des données, la réplication permet de multiplier le nombre de copies dites répliques d'un fragment à travers plusieurs sites.

Une autre approche similaire à la réplication, qui consiste à placer la donnée à l'endroit où elle est demandée pour garantir sa disponibilité et éviter le transfert pour les prochaines requêtes qui la traitent, correspond au mécanisme de caching.

Afin de garantir la cohérence des données répliquées ou du cache par rapport à la source, une stratégie de gestions des mises à jour est utilisée.
Lorsque les capacités de stockage d'un site sont limitées pour contenir de nouvelles données issues du processus de cache ou de réplication, une stratégie de remplacement qui décide des fragments à supprimer ou à déplacer est utilisée.

2.1.1.1 Fragmentation de données

Avec l'automatisation des mécanismes de collection de données, le volume des données hébergées dans les entrepôts ne cesse de s'accroitre, de même que la complexité des requêtes. Les traitements centralisés de ces données peuvent impacter négativement les performances du système. En effet, un système de traitement centralisé concentre les requêtes sur un seul site qui est limité en ressources de stockage et de calcul. Ce fonctionnement peut engendrer des congestions qui freinent le système pour le passage à l'échelle des données et des traitements. Afin d'optimiser l'utilisation des ressources, un mécanisme de fragmentation des relations volumineuses est utilisé. La fragmentation ou partitionnement permet de paralléliser les traitements des requêtes impliquant des portions de données disjointes tout en économisant les ressources de stockage. Il consiste à découper les données en portions moins volumineuses et pouvant être placées chacune à l'endroit où elle est exploitée. De ce fait, il permet non seulement de paralléliser des requêtes indépendantes (on parle de parallélisme inter-requêtes), mais aussi une requête accédant à des portions de données disjointes, peut être découpé en plusieurs sous-traitements (moins gourmands en ressources). Ces sous-traitements peuvent être exécutés en parallèle à travers les sites qui disposent de fragments impliqués (on parle de parallélisme intra-requête).

La fragmentation a largement été étudiée dans le contexte des bases de données réparties [OV11, BBRW09]. Elle consiste à diviser un ensemble de données en plusieurs sous-ensembles qui

peuvent être non disjoints. La fragmentation intervient principalement dans la mise en place d'un système réparti pour la gestion de données tels que les Système de Gestion de Base de données Répartis (SGBDR), les systèmes pair-à-pair, les grilles de données ou encore les systèmes de cloud offrant les services de base de données (DBaaS).

2.1.1.2 Allocation et réplication

La stratégie de placement des fragments à travers les sites d'un système réparti a pour objectif de garantir la disponibilité des fragments pour les requêtes qui interrogent ceux-ci (les fragments) tout en réduisant les coûts de transfert entre les sites. En outre, elle vise à paralléliser les traitements des requêtes indépendantes (qui portent sur des fragments disjoints). Pour ceci, cette stratégie fait intervenir deux concepts proches mais différents : l'allocation et la réplication.

Le processus d'allocation des fragments est l'étape qui suit la phase de fragmentation. Il consiste à allouer chaque fragment en un site du système réparti. Le schéma d'allocation désigne la façon dont les fragments sont alloués aux sites. Il est généralement statique dans les systèmes de gestion de données classiques. Cependant, le schéma de placement peut être dynamique, lorsqu'il adopte une stratégie de replacement de fragments au cours de l'exploitation du système. Le problème principal du placement de données est de trouver le schéma de placement qui optimise les performances du système avec un coût acceptable.

La réplication consiste à dupliquer un fragment en plusieurs copies qui seront placées en des sites différents. La première copie qui a été dupliquée pour créer les autres copies est appelée copie maître et les autres sont dites copies secondaires ou esclaves. Chaque copie du même fragment est appelée réplique. Le degré de réplication d'un fragment est le nombre de répliques du fragment dans le système.

L'objectif de la réplication est de garantir la disponibilité d'un fragment afin de favoriser les traitements parallèles sur celui-ci et donc de réduire les congestions des requêtes qui interrogent le fragment. En effet, lorsqu'un fragment est très sollicité, la réplication de celui-ci à travers plusieurs sites permet de traiter de façon simultanée différentes requêtes qui invoquent le fragment afin d'équilibrer la charge à travers les sites et de réduire la latence des requêtes. Les problèmes majeurs dus à la réplication sont la gestion de la cohérence des différentes répliques, le contrôle du degré de réplication et la gestion de l'espace de stockage. Le contrôle du degré de réplication a son importance pour la gestion de la cohérence des répliques et dépend de la nature des traitements dominants du système [GSN09]. La plupart des systèmes comme Hadoop [F.c],et ses dérivées [ABPH$^+$10, BPASP11, T$^+$10, SKRC10], limite leur degré de réplication à une valeur fixe et commune à tous les fragments d'un fichier de données. Pour des systèmes qui impliquent des mises à jour fréquentes, un degré de réplication élevé engendrerait des coûts de communication importants pour garantir la cohérence des données. Par contre, lorsque les traitements sont

orientés analytique (lecture seule), c'est-à-dire lorsque les requêtes de mises à jour sont rares, alors un degré de réplication élevé peut augmenter les performances du système lorsque les sites ont des capacités de stockage adéquates. La gestion de la cohérence des répliques se fait grâce à une stratégie de gestion des mises à jour qui peut être synchrone ou asynchrone. La gestion synchrone des propage immédiatement les mises à jour apportée sur un fragment vers toutes ses répliques. Avec ce mécanisme, les répliques d'un fragment sont identiques à tout moment. Ce mode de gestion des mises à jour est plus adapté aux applications transactionnelles qui exigent une forte cohérence des données. La gestion asynchrone quant à elle, propage plutard les mises à jour d'un fragment vers ses répliques. Elle est adaptée aux applications analytiques où les modifications sont rares.

La stratégie de réplication des fragments peut être statique ou dynamique. Elle est dite statique lorsque chaque réplique est placée en permanence sur un site. Dans ce cas, le schéma de réplication d'un fragment ne change pas au cours du temps. Avec la stratégie de réplication dynamique, le nombre de répliques d'un fragment ou l'emplacement d'une réplique peuvent varier au cours de l'utilisation du système. On peut ainsi créer, supprimer ou déplacer une réplique en fonction de la politique de réplication définie. Un autre mécanisme qui est similaire de la réplication est le caching de données qui vise à rapprocher les résultats des requêtes aux sources de celles, les utilisateurs.

2.1.1.3 Caching de données

Le cache de données permet de stocker les résultats de requêtes dans la machine de l'utilisateur. De ce fait, lorsque l'utilisateur soumet ultérieurement la même requête, le résultat sera rapidement à sa disposition. L'avantage de la cache des résultats est qu'elle réduit à la fois le transfert de données entre le serveur et l'utilisateur mais également le coût d'évaluation de la requête sur le serveur pour le traitement itératif d'une série de requêtes redondantes. De ce fait, le caching permet de garantir la disponibilité des données invoquées par des requêtes fréquentes. Cependant, lorsque les données relatives à ces caches sont modifiées au serveur, alors les résultats des requêtes ne seront plus cohérents. Un deuxième problème est que lorsque les requêtes de l'utilisateur ne sont pas redondantes et qu'elles n'ont les mêmes résultats, alors ce type de cache peut générer un coût de stockage élevé. En outre, dans ce cas, il ne réduit pas les transferts ou les coûts d'évaluation des requêtes. Un autre problème est que le caching génèrent un coût de stockage qui est généralement proportionnel aux nombre de requêtes qui exploitent les données. Ceci peut poser un réel problème pour le passage à l'échelle des données et des applications. Un mécanisme de remplacement est souvent adopté pour adapter dynamiquement le contenu des sites aux applications.

2.1.1.4 Stratégie de remplacement de données

La stratégie de remplacement de données se résume en un mécanisme de mises à jour qui permet d'insérer de nouvelles données dans la base locale ou la cache lorsque l'espace de stockage disponible est atteint. Pour cela, des données doivent être enlevées de la base locale. Cet enlèvement peut être une suppression ou un déplacement vers une base distante. Le choix des données à conserver ou à enlever est déterminé par un algorithme de remplacement. Plusieurs algorithmes peuvent être envisagés dans ce sens.

- **Least Recently Used (LRU) :** cet algorithme enlève de la base les données les moins récemment utilisées. Le principe est de conserver les données en cours d'utilisation et récemment traitées, qui auraient plus de chance d'être réutilisées dans un futur proche. Il est possible d'implémenter cet algorithme en gardant les traces (dates) d'accès aux différentes unités de données. En cas de remplacement les données ayant les dates d'accès les plus récentes sont conservées en priorité.
- **First In First Out (FIFO) :** cet algorithme enlève de la base, les données les plus vieilles pour conserver les données nouvellement insérées (plus jeunes). Pour implémenter cet algorithme, on peut journaliser la date d'insertion de chaque unité de donnée.
- **Least Frequently Used (LFU) :** consiste à supprimer les données les moins fréquemment utilisées afin de conserver les données les plus populaires. Pour implémenter LFU on trace la fréquence d'accès à chaque unité de donnée par un compteur qui est incrémenté à chaque demande de la donnée. Ces fréquences d'accès peuvent aussi être mesurées sur une période glissante.
- **Algorithme aléatoire :** cet algorithme consiste à choisir aléatoirement les données à enlever de la base. De ce fait, n'importe quelle donnée peut être enlevée.

On peut aussi noter des variantes de ces algorithmes ou des combinaisons. Toutefois, la stratégie de remplacement doit tenir compte de la nature des traitements effectués sur les données pour optimiser les performances. En plus, même si les capacités de stockage sont limitées, une bonne stratégie de remplacement doit garantir la disponibilité des données.

2.1.2 Stratégie de répartition de données

2.1.2.1 Revue de la littérature

Une bonne stratégie de répartition de données dans un environnement réparti est primordiale pour la garantie de performance et du passage à l'échelle. La stratégie de répartition aboutit à un schéma de placement des données à travers les sites qui constituent le système. Plusieurs travaux ont été menés dans le sens de la répartition de données dans les systèmes répartis [Gop12, ZBL12, NB11, PPR+09, FS12, SPTB14, KPX+11, HAD09, SCSA08, Wie14, TMX+13, RF01].

La plupart des solutions se sont concentrées sur les caractéristiques (ou attributs) des données pour définir les classes d'accès. Conscients de l'importance de la nature des traitements exécutés sur les données, d'autres travaux ont tenté de compléter les solutions antérieures avec de nouvelles techniques de répartition en considérant les spécificités (analytiques ou transactionnelles) des requêtes. Il importe de noter que la plupart des travaux menés dans le sens de la répartition de données aboutit à un schéma de placement statique qui est déterminé à la mise en place du système et reste inchangé pendant la phase d'exploitation du système.

Avec l'avènement des nouvelles architectures réparties à très large échelle tel que le cloud [AEADE11, Aba09, FS12] qui sont caractérisées par l'hétérogénéité de leurs composants et leurs charges dynamiques, plusieurs études [FS12, AEADE11] se sont intéressées sur des mécanismes de répartition de données qui s'adapteraient à ces nouvelles architectures. En effet, les techniques classiques de répartition de données dans une architecture répartie, ne sont plus applicables au contexte des environnements répartis à très large échelle [AEADE11]. Ces derniers sont caractérisés par un nombre élevé de sites qui ont des capacités de stockage et de calcul hétérogènes [Aba09]. En outre, les capacités des liens de communication entre les nombreux sites sont hétérogènes [Aba09]. Lorsque le système est en cours d'exploitation, les charges fluctuent d'un site à un autre et sur un même site au cours du temps [AEADE11]. A ces caractéristiques s'ajoutent les risques élevés de pannes de sites ou les problèmes techniques posés par l'intégration de nouvelles machines.

Par ailleurs, la nature *Big Data* [tE14] des données hébergées dans de tels systèmes rend leur gestion plus complexe par rapport à des données relationnelles. En effet, les Big Data sont caractérisées par leur quantité énorme, leur taux d'accroissement rapide et leurs variétés [FS12]. La répartition de données Big Data nécessite un mécanisme de placement des données qui adapte les ressources de stockage et de calcul disponibles au volume important de données pour garantir leur disponibilité. En effet, la disponibilité est une exigence des systèmes comme le cloud pour garantir un accès continu au service [AEADE11, FS12]. En outre, le mécanisme de gestion de données doit aussi garantir le passage à l'échelle rapide des données et des applications qui les exploitent tout en étant flexible pour l'intégration rapide de nouvelles variétés de données.

Les limites des systèmes de base de données relationnelles dans le contexte des architectures à large échelle ont propulsé de nouvelles technologies de gestion de données dans de tels systèmes. Ces solutions peuvent être regroupées en deux familles :

i) les solutions comme HadoopDB [ABPH+10, BPASP11], SCOPE [CJL+08], qui utilisent les systèmes de gestion de données parallèles classiques en leurs ajoutant une couche pour optimiser les communications entre les sites. Ainsi, elles bénéficient des performances et des fonctionnalités des systèmes de bases de données relationnels tout en s'adaptant au contexte du large échelle,

ii) les solutions NoSQL (pour Not only SQL) comme Hadoop[F.c],Pig[F.b], Hive[T+10], Shark [XRZ+13], Dryad [IBY+07], CoHadoop [ETO+11], qui implémentent ou étendent le modèle MapReduce [DG08] basé sur une structure clé-valeur des données. Ces solutions utilisent un système de gestion de fichiers distribué comme Google File System (GFS)[GGL03], BigTable [C+06], Hadoop Distributed File System (HDFS) [SKRC10] ou Cosmos Storage System [CJL+08]. Pour ce qui concerne la répartition des données à travers le système, ces solutions utilisent généralement un mécanisme de partitionnement basé sur un modèle de hachage qui, en fonction de la clé identifiant la donnée, détermine le site qui doit le stocker. Une donnée peut être répliquée avec un degré fixe de réplication. Il importe de noter que pour la plupart de ces systèmes, le schéma de placement d'une donnée est statique.

D'autres travaux concernant de nouvelles stratégies de répartition de données massives dans les environnements à large échelle ont été menés [KKAP96, LA00, Gop12, ZBL12, NB11, FS12, KPX+11, HAD09, Wie14, TMX+13, CJP+11, KGD11]. Ces recherches prennent en compte certains paramètres qui caractérisent ces systèmes et les traitements qui y sont effectués [AEADE11, Aba09]. Ces paramètres concernent la différence entre les capacités (de calcul et de stockage) des machines, des liens de communication, les charges des différents sites, la nature des données et les propriétés des opérations des requêtes.

2.1.2.2 Quelques travaux spécifiques sur la répartition de données

Nous détaillons quelques travaux spécifiques qui traitent la répartition des données à large échelle.

– **CoHadoop (2011) [ETO+11].** L'objectif de CoHadoop est de permettre aux applications de contrôler le placement des données qui n'est pas contrôlé avec Hadoop. CoHadoop est une extension de Hadoop pour colocaliser les fichiers de données liées qui peuvent être traités ensemble pour améliorer les performances notamment pour les jointures de ces fichiers. Cette approche attribue à chaque nouveau fichier une propriété appelée *locator* qui indique son futur emplacement. Cette propriété dépend des applications. Un nouveau fichier est placé au site qui héberge les fichiers antérieurs de même *locator* lorsqu'ils existent. Sinon, le placement par défaut de Hadoop est utilisé. De ce fait, les fichiers de même *locator* sont colocalisés. De plus, CoHadoop colocalise les répliques des fichiers de même *locator*. La colocalisation des données de CoHadoop permet de réduire le coût des requêtes de jointure lorsque les données impliquées se trouvent sur le même site en réduisant le coût de transfert des données. De ce fait, la colocalisation permet de réduire aussi les risques de blocage de la requête.
  Cependant, CoHadoop nécessite de déterminer à l'avance quels sont les fichiers à colocaliser. CoHadoop n'apporte pas de bénéfice pour un traitement sur deux fichiers ayant des localisations différentes.

De plus, la quantité de données sollicitées par une requête peut dépasser l'espace de stockage d'un site. Or, cette approche vise à optimiser le traitement local d'une requête et, comme Hadoop, ne supporte pas le traitement réparti d'une requête de jointure. En outre, CoHadoop agit sur les fichiers qui viennent d'être intégrés au système et cette colocalisation n'est pas adaptée dynamiquement pour repartitionner les données qui sont déjà dans le système en fonction du changement des accès et des charges disparates et fluctuantes des sites.

– **Travaux de Rimma Nehme (2011) [NB11].** Dans ce travail, Nehme montre que dans les environnements de traitements parallèles massifs, les données sont partitionnées sur plusieurs nœuds de calcul. Ce partitionnement des données aboutit à une amélioration des performances spectaculaires lors de l'exécution de requêtes en parallèle. Nehme constate que pour évaluer certains opérateurs, des données nécessitent d'être réparties à nouveau à travers les sites. Il considère aussi que le transfert des données est plus coûteux. Il propose ainsi un mécanisme de partitionnement (appelé Advisor) qui minimise les transferts entre les sites pour une charge donnée. Cette solution intègre le processus de repartitionnement dans l'optimiseur de requêtes. Particulièrement, le module de partitionnement exploite le modèle de coût de l'optimiseur ainsi que la structure de données pour trouver la meilleure configuration de partitionnement possible. Le modèle de coût est déterminé en fonction des charges des différents sites, des coûts de transfert et des opérations de la requête, et de la structure des données.
  L'avantage de cette approche est l'adaptation du schéma de placement des données en fonction de la structure des données et de l'état courant des sites. Ce qui joue un rôle important dans le contexte des environnements répartis à large échelle pour garantir le passage à l'échelle.
  Cependant par rapport à l'approche de minimiser le coût de transfert de chaque requête, il peut être intéressant d'accepter de transférer une donnée même si c'est coûteux afin de favoriser sa disponibilité pour les prochaines requêtes qui l'invoquent. Si l'objectif n'est pas de minimiser les transferts mais de respecter une contrainte de temps de réponse, alors transférer une donnée vers un site moins chargé lorsqu'il n'existe aucune possibilité pour respecter la contrainte, peut être bénéfique pour la requête en cours et les requêtes ultérieures.

– **Farah Habib Chanchary et al (2012) [FS12].** Dans ce travail Chanchary propose un mécanisme de migration de données dans le cloud en considérant les exigences des utilisateurs de ce type de système que sont le débit transactionnel, l'évolutivité, la latence, la disponibilité et la fiabilité. Du coté des fournisseurs, l'intérêt se focalise sur la manipulation de la charge, l'allocation effective des ressources, la sécurité des données etc. Chanchary et al abordent la migration de données des grandes entreprises vers le cloud de façon efficace. Pour cela, ils proposent une architecture de migration avec un gestionnaire du partitionnement des données (Data Partitioning Manager). Ce module instancie le modèle de partitionnement de *Relational cloud* [CJP+11] qui répartit les données d'un nœud unique vers plusieurs sites en fonction de la charge pour aboutir à un schéma de placement qui équilibre la charge entre les sites. Le modèle de partitionnement de Relational Cloud utilise les informations des charges courantes des sites. Pour cela, il analyse périodiquement les traces d'exécution de chaque

requête pour identifier les tuples qui sont accédés ensemble pour des requêtes individuelles. De cette analyse, il déduit un graphe où chaque nœud représente un ensemble de tuples et les arcs correspondent à des mouvements des tuples d'une même requête. Pour améliorer les performances, cet algorithme partitionne la base entière sur les attributs interrogés. Les fragments sont ensuite répartis à travers les sites qui reçoivent des demandes régulières.
L'objectif de partitionner dynamiquement les données en fonction des classes d'accès permet d'adapter le schéma de placement des données en fonction des requêtes. Ceci peut jouer un rôle important pour les performances du système. En outre, l'approche de regrouper les données de la même requête en un seul site peut améliorer les performances du système en favorisant le traitement local de la requête et réduisant ainsi les transferts et les risques de blocage qu'aurait induit un traitement réparti.
Cependant, un traitement local suppose que toutes les données sont dans une seule base. Or, la taille d'une base est limitée. Lorsque les données nécessaires au traitement de la requête excèdent la taille de la base, alors l'exécution de la requête devient impossible. Dans ce contexte il serait intéressant de répartir les données de la requête à travers un nombre minimal de sites qui permettrait le traitement de la requête. Le choix des sites qui interviendraient pourrait être affiné par une contrainte de temps de réponse borné et un modèle de coût dynamique qui tiendrait compte des coûts de transfert, d'attentes dues aux charges, et des opérations de la requête.

– **Travaux de Gope (2012) [Gop12].** Gobe identifie les problèmes de l'allocation statique des données pour les performances d'un système dynamique. En effet, selon Gope, dans un environnement statique où les probabilités d'accès des nœuds vers les fragments ne changent pas, une allocation statique de fragments fournit une meilleure solution. Toutefois, dans un environnement dynamique où ces probabilités changent au cours du temps, la solution d'allocation statique dégraderait les performances de la base de données. Dans ses travaux, Gope propose un algorithme d'allocation dynamique de données dans un système de base de données non-répliquées. Cet algorithme consiste à compter le nombre de demandes de chaque fragment soumises à chaque site du système et transférer le fragment vers le site ayant reçu le plus grand nombre de demandes. Ce transfert est effectué quand le nombre de sollicitations du fragment est élevé. L'algorithme NNA (pour Near-Neighbor Algorithm) conserve le même principe pour prendre la décision du transfert du fragment. Cependant, la destination peut être différente. Dans l'algorithme NNA, le destinataire du fragment est le nœud le plus proche de la source qui se trouve sur le chemin menant vers le site ayant reçu le plus grand nombre de demandes du fragment. L'objectif de cette approche de Gope est de réduire le temps de réponse en réduisant le coût des transferts, car cela rapproche petit à petit le fragment du site recevant le plus de requêtes concernant ce fragment.
Cette stratégie d'allocation dynamique peut être intéressante lorsque le coût de transfert est élevé et dépend de la distance qui sépare la source et la destination.
Cependant, l'algorithme NNA ne considère pas les charges de la source ou du destinataire

dans la prise de décision de déplacer le fragment, alors que ceci permet d'optimiser le temps de réponse de la requête. En effet, lorsque les utilisateurs d'un site sont largement plus actifs, alors NNA tend à regrouper les données vers ce site. Ceci réduit certes les transferts mais engendre une surcharge du site au moment où d'autres machines seraient libres. On peut conclure que, cette approche de Gope converge à une situation où peu de sites du système sont utilisés. En outre, en supposant que la répartition des utilisateurs à travers les sites est uniforme et que chaque utilisateur peut accéder à n'importe quelle donnée, alors les déplacements d'un même fragment peuvent être récurrents. Ces déplacement récurrents s'expliquent surtout par le fait qu'un seul site dispose du fragment et donc du non-support de la réplication. En effet, la possibilité de répliquer le fragment vers la destination tout en gardant une copie à la source permettrait de garantir plus de disponibilité.

– **Travaux de Jingren Zhou (2012) [ZBL12].** Dans ce travail, Zhou considère que le processus de réorganisation des données est plus coûteux dans le contexte des traitements répartis massifs. Il montre que les solutions qui existaient, étaient rigides en termes de fonctionnalités et que leurs manipulations n'étaient pas naturelles pour les utilisateurs. C'est ainsi qu'il propose des techniques avancées de partitionnement qui dépendent des propriétés des données et qui optimisent les mouvements des données à travers le réseau. En conséquence, elles seront en mesure d'éviter les repartitionnements inutiles des données. Les techniques sont applicables à la fois à la compilation et à l'exploitation du système.

  L'avantage de cette solution est que le schéma de placement des données est dynamique et que sa gestion est guidée par les propriétés des données impliquées dans les requêtes ainsi que la nature des traitements courants du système. En outre, la réorganisation des données est complétée par l'optimiseur de requêtes qui évalue les coûts de différents plans d'exécution de la requête pour décider des données à réorganiser et de leurs éventuelles destinations. Ce qui peut être très important dans le contexte des environnements répartis à très large échelle.

  Cependant, dans ces types de système la prise en compte des charges dynamiques des nœuds est essentielle dans l'optimisation de la requête vu la concurrence d'accès aux ressources. Ce paramètre n'est pas intégré dans l'estimation du coût de la requête.

– **Travaux de Jiexing Li et al (2014) [JJR14].** Li montre les limites du partitionnement uniforme de données dans le contexte des environnements répartis. En effet pour Li, la stratégie de partitionnement uniforme peut engendrer la surcharge des sites de faible capacité et une sous-utilisation des sites qui disposent largement de ressources. Pour lui, lorsque le temps d'une requête parallèle est déterminé par rapport à une machine lente, alors on peut assister à une dégradation des performances pour la requête.

  Pour résoudre ce problème de performance de la requête lié à l'hétérogénéité des capacités des sites, Li propose un mécanisme de partitionnement qui alloue plus de données aux sites les plus performants et moins aux sites les moins performants. Pour cela, il propose trois approches, *uniform, delete* et *optimal*. L'objectif de son principe est de partir de l'approche simple *uniforme* pour aboutir à l'approche *optimal* en passant par une approche intermédiaire *delete*.

L'approche *uniform* correspond au mécanisme de partitionnement classique de données. L'approche *delete* supprime les machines de faibles performances pour placer les données à travers les autres sites. L'approche *optimal* répartit les données à travers les sites d'une façon à minimiser le temps d'exécution de la charge globale. Pour connaitre les performances des différents sites, Li utilise une série de requêtes de calibrage qu'il exécute sur chaque site pour quantifier les performances de celui-ci.

L'avantage de cette approche est qu'elle permet d'allouer plus de traitements aux sites les plus performants tout en évitant les transferts de données. En outre, elle permet d'équilibrer la charge en fonction des performances des sites et de la localisation des données impliquées dans les requêtes. En d'autres termes, plus un site est performant, plus il dispose de données et donc plus il est sollicité.

Cependant, il importe de noter que le placement d'une quantité énorme de données sur un site performant ne garantit pas toujours un équilibrage de la charge. En effet, les données du site performant peuvent ne pas être utiles pour les utilisateurs qui verraient ainsi leurs requêtes redirigées vers des sites ayant des performances faibles. Ces derniers peuvent facilement être surchargés vu la quantité de traitements qui leur sont soumis et la médiocrité de leur capacité de calcul. En outre, lorsque les données du site performant sont très sollicitées, une requête peut aboutir à une latence considérable avant d'être traitée, alors qu'en même temps, un site de performance moyenne serait libre. Ceci n'optimise pas l'utilisation globale des ressources. Et pourtant une réplication de certains fragments vers d'autres sites permettrait de paralléliser certains traitements. Les principales limites de cette solution peuvent se résumer à la non-réplication de données et surtout à l'inadaptation dynamique du schéma de placement en fonction des charges des différents sites.

- **Travaux de Ranganathan et Foster(2001) [RF01].** Ranganathan et Foster proposent six stratégies alternatives de réplication dynamiques de données dans un système réparti. Cette solution de caching de données suppose que toutes les données sont initialement en un seul site appelé *root* qui est le nœud racine d'une organisation hiérarchique.
  - *Stratégie 1 : No caching no Replication :* dans cette stratégie, toute donnée est accessible uniquement à partir de la racine et aucune réplication ou caching n'est effectuée sur le client ou sur les serveurs intermédiaires.
  - *Stratégie 2 : Best Caching :* cette approche identifie pour chaque fichier le meilleur client dans chaque nœud par intervalle de temps. Le meilleur client pour un fichier est celui qui a génèré le plus grand nombre de requêtes invoquant le fichier. Ce nombre doit dépasser un certain seuil fixé à l'avance. Dans ce cas, une réplique est ensuite créée et envoyée au meilleur client.
  - *Stratégie 3 : Cascading Replication :* fonctionne avec une organisation arborescente dans laquelle les feuilles représentent les clients et la racine correspond au serveur unique contenant initialement toutes les données. Lorsque le nombre de demandes d'un fichier atteint un seuil à la racine, alors une réplique est créée et placée au niveau suivant sur le serveur qui

se trouve sur le chemin qui mène vers le meilleur client. Ainsi le nouveau site est un ancêtre du meilleur client. Une fois que le nombre de demandes au niveau 2 pour le fichier dépasse le seuil, alors le fichier est répliqué immédiatement au niveau inférieur et ainsi de suite. On constate qu'avec cette stratégie, un fichier populaire peut rapidement être répliqué sur le meilleur client.

- *Stratégie 4 : Plain Caching :* à chaque fois qu'un client demande un fichier, il le copie localement. Lorsque la capacité de stockage est atteinte, alors les fichiers antérieurs sont remplacés par les fichiers des nouvelles demandes.
- *Strategy 5 : Caching plus Cascading Replication :* elle combine les approches 3 et 4. Le client cache localement les fichiers. Périodiquement, le serveur identifie les fichiers populaires et les propage dans la hiérarchie arborescente où les clients sont les feuilles et n'importe quel nœud peut être un serveur. Donc un client peut être serveur et servir à ses frères.
- *Strategy 6 : Fast Spread :* dans cette approche, une réplique est créée sur chaque nœud qui se trouve sur le chemin menant vers le client lorsque celui-ci demande le fichier.

Conscient du coût considérable généré par les solutions qu'il a proposées, Ranganathan complète sa proposition avec une stratégie de remplacement de fichier qui retient uniquement les fichiers populaires lorsque de nouveaux fichiers doivent être stockés. Cette nouvelle stratégie prend en compte aussi l'âge des répliques dans la décision de suppression. En effet, un fichier qui vient d'être répliqué n'est pas immédiatement populaire mais il pourrait le devenir rapidement dans le futur. Ainsi lorsque deux fichiers ont la même popularité, alors le plus ancien est supprimé. Les traces de popularité des fichiers sont effacées de façon périodique pour capturer le comportement dynamique des utilisateurs. Cette étude lui permet de ressortir deux stratégies qui offrent de meilleures performances pour la réplication dynamique de données en fonction des classes d'accès. En effet, pour Ranganathan, Fast Spread (Stratégie 6) est plus efficace lorsque les classes de requêtes sont aléatoires. Cascading (stratégie 5) est meilleure quand il y a une petite localité des classes aux fichiers.

Ces différentes stratégies permettent de garantir la disponibilité des données en les rapprochant des clients et multipliant leurs copies en fonction de leur popularité. En outre, la stratégie de remplacement permet de préserver les ressources de stockage tout en garantissant la disponibilité des données populaires. Ce qui permet d'optimiser les performances du système en réduisant les transferts.

Cependant, la prise en compte des charges des sites qui accueillent les répliques pourrait jouer un rôle important dans l'équilibre de la charge à travers le système. En effet, lorsque les classes d'accès sont localisées à travers un nombre limité de sites, les stratégies proposées regroupent davantage les données sur ces sites au moment où leurs voisins ne sont inutilisés ni pour stocker des données, ni pour traiter des requêtes. Ceci peut être un frein pour la garantie du passage à l'échelle.

- **Travaux de d'Orazio et al (2010) [dRL10]**D'Orazia et al proposent une solution qui adapte dynamiquement la cache pour des applications hétérogènes en fonction du contexte

d'usage. Pour cela, ils considèrent à la fois des techniques de caches sémantiques et des techniques de caches coopératives. Pour ce qui concerne la cache sémantique, les auteurs considèrent les caches basées sur les résultats, les caches basées sur les prédicats et une approche dual cache qu'ils ont proposée. Le caching basé sur les résultats associe à chaque entrée de la cache, une requête, celle qui était à l'origine de sa création. Les auteurs considèrent le pourcentage d'utilisation de chaque entrée dans le processus de remplacement. Le caching basé sur les prédicats des requêtes permet de partager les entrées de la cache sans nécessité de réplication pour des requêtes différentes mais qui ont des prédicats en commun. La gestion du stockage est liée à la granularité des objets. Pour ce qui concerne dual cache, elle combine des caches d'objets et des caches de requêtes pour améliorer l'évaluation de requêtes. En ce qui concerne les caches coopératives, les auteurs considèrent la résolution verticale et la résolution horizontale. La résolution verticale permet de rapprocher davantage la donnée aux utilisateurs qui s'intéressent. La résolution horizontale permet de partager les contenus des caches de plusieurs sites qui sont au même niveau par rapport aux serveurs. Cette solution adapte dynamiquement l'utilisation des caches au contexte hétérogène des applications qui exploitent les données en combinant plusieurs types de caches selon le contexte d'usage des données. Cependant, même si elle permet de partager les contenus des caches qui se trouvent sur plusieurs sites, elle n'exploite pas la possibilité de traitement réparti d'une requête qui permettrait de bénéficiait du parallélisme surtout pour des requêtes longues. En outre, elle ne considère pas la possibilité de créer dynamiquement de nouvelles répliques pour une donnée en fonction de son utilisation et de la charge des sites qui en disposent. Ce qui peut poser un problème d'adaptabilité du placement des données dans les caches pour le passage à l'échelle des applications qui les exploitent.

2.1.2.3 Synthèse

Nous récapitulons dans le tableau ci-dessous les fonctionnements des approches de partitionnement de données que nous venons de présenter. Ces études nous permettent de distinguer deux états du schéma de placement : dynamique et statique. Nous présentons brièvement les fonctionnalités qui sont utilisées dans les solutions de répartition dynamique de données.

- **Schéma de placement dynamique.** Les solutions de partitionnement dynamiques [RF01, NB11, FS12, Gop12, ZBL12, ETO$^+$11] adaptent leurs schémas de placement en fonction du contexte dynamique des environnements répartis à large échelle.
- **Paramètres d'optimisation.** Pour ces solutions, la technique de partitionnement est plus ou moins liée au processus d'optimisation de requêtes. Ce qui s'avère primordial puisque les performances du système dépendent directement du schéma de placement des données. Une bonne stratégie d'optimisation de requêtes dans les systèmes répartis doit tenir compte des paramètres susceptibles d'affecter le coût de la requête. Parmi ces paramètres, on peut citer

la disponibilité des données et des ressources de calcul, le coût des éventuels transferts, la nature des traitements. La disponibilité des ressources de calcul d'un site dépend de son occupation ou charge. La charge d'un site correspond à la somme des coûts de l'ensemble des traitements en cours d'exécution et en attente sur ce site. Le coût d'un traitement local dépend de la nature des opérations impliquées et des performances du site. La réduction des coûts de transfert peut avoir un rôle important pour améliorer les performances. C'est ainsi que certaines solutions visent à optimiser les performances en minimisant le coût des transferts [NB11]. La disponibilité d'une donnée est fortement liée à son degré de réplication et aux charges des sites qui en disposent comme réplique. Le degré de réplication d'une donnée peut être fixe ; lorsqu'il change au cours de l'exploitation du système, ce degré est dit dynamique.

- **Réplication dynamique.** Le mécanisme de réplication dynamique pour une donnée permet d'adapter sa disponibilité en fonction de la quantité de requêtes qui la traitent au cours du temps. Le mécanisme de réplication dynamique permet de dupliquer davantage les données les plus sollicitées. De ce fait, les données les moins utilisées sont aussi moins répliquées. Ce qui permet d'éviter les coûts de stockage inutile de certaines données moins populaires qu'aurait induit un degré de réplication uniforme élevé. Toutefois, la réplication dynamique peut engendrer un surcoût de stockage, or les ressources sont limitées dans ce sens. Ainsi pour garantir à la fois le passage à l'échelle et la disponibilité des données, un mécanisme de remplacement des données obsolètes est essentiel.
- **Stratégie de remplacement.** La stratégie de remplacement a pour objectif de garantir la disponibilité des données sollicitées par les requêtes tout en préservant les ressources de stockage. De ce fait, une solution de placement de données avec le support de la réplication dynamique doit disposer d'une stratégie de remplacement pour garantir le passage à l'échelle.
- **Temps de réponse borné.** Le choix de borner les temps de réponse des requêtes peut jouer un rôle important sur le mécanisme de partitionnement des données. En effet, un temps de réponse borné permettrait d'éviter des transferts qui visent à minimiser le temps de réponse de la requête. En outre, il permet d'équilibrer la charge à travers les sites en transférant les données de la requête qui se trouvent sur un site surchargé (qui ne pourrait pas respecter la contrainte de temps de réponse borné) vers un autre site qui respecte la contrainte de temps de réponse borné.

2.1.2.4 Positionnement de notre solution

Le tableau 2.1 nous permet de constater les limites des travaux précédents pour s'adapter au contexte des environnements répartis à large échelle. Ils n'intègrent pas tous les paramètres que nous considérons indispensables à un modèle de placement de données.

Travaux	Placement dynamique	Paramètre d'optimisation	Réplication dynamique	Stratégie de remplacement	Temps de réponse borné
Rimma Nehme (2011)	Oui	– Coût de transfert – Structure des données – Charges des sites	Oui	Non	Non
Farah Habib Chanchary et al (2012)	Oui	– Charge des sites – Strucuture des données	Oui	Non	Non
Gope (2012)	Oui	– Coût de transfert	Non	Non	Non
Jingren Zhou (2012)	Oui	– Coût des transferts – Structure des données – Propriété des requêtes	Oui	Non	Non
Jiexing Li et al (2014)	Non	– Performances des sites	Non	Non	Non
CoHadoop (2011)	Oui	– Propriétés des requêtes – Coût des transferts	Non	Oui	Non
Ranganathan et Foster(2001)	Oui	– Propriétés des requêtes	Oui	Oui	Non
D'Orazio et al (2010) [dRL10]	Oui	– Propriétés des requêtes – Propriétés des données – Coût des transferts	Oui	Oui	Non

TABLE 2.1 – Travaux spécifiques de répartition de données

Travaux	**Placement dynamique**	**Paramètre d'optimisation**	**Replication dynamique**	**Stratégie de remplacement**	**Temps de réponse borné**
Notre approche	Oui	– Coût de transfert – Structure des données – Charges des sites – Performances des sites – Nature des requêtes	Oui	Oui	Oui

TABLE 2.2 – Positionnement de notre approche

Nous approche de répartition dynamique, qui fragmente les données en fonction de leur structure et des prédicats des requêtes, est guidée par l'optimisation de requêtes, dont l'objectif est de respecter une contrainte de temps de réponse borné.

La stratégie d'optimisation est basée sur un modèle de coût qui est évalué en fonction de plusieurs paramètres susceptibles d'affecter le coût d'une requête dans le contexte d'un environnement réparti hétérogène à large échelle. En effet, le coût d'une requête est évalué en fonction des charges courantes des sites, de leurs performances, des coûts de transferts de la nature des opérations des requêtes et de la contrainte de temps de réponse borné. Les transferts concernent principalement les réplications dynamiques de fragments vers des sites moins chargés pour garantir leurs disponibilités. Ce qui favorise davantage le respect de la contrainte de temps de réponse. Afin de sauvegarder les ressources de stockage tout en garantissant la disponibilité des données, nous proposons une stratégie de remplacement qui est appliquée à chaque base lorsque sa taille autorisée est dépassée. Notre stratégie de remplacement enlève les données les moins populaires de la base en les déplaçant vers les voisins ou les supprimant de la base locale. Elle supprime une donnée lorsque celle-ci dispose d'une copie dans une base voisine. Ce qui permet de conserver toutes les données à travers le système. Le fonctionnement de notre approche est différent de celui des approches de caching données coopératives classiques dans la mesure où il adapte dynamiquement le placement de données en fonction de plusieurs paramètres qui peuvent impacter le coût d'exécution de la requête. Notre approche considère ces paramètres dans l'optimisation de la requête qui guide le placement des données et supporte la possibilité de traitement réparti d'une requête pour bénéficier du parallélisme intra-requête en plus du parallélisme inter-requêtes. Dans le tableau 2.2, nous montrons le fonctionnement de notre approche par rapport aux travaux présentés en haut.

2.2 Optimisation de requêtes à large échelle

Le traitement de requêtes dans les environnements répartis à large échelle est assez complexe, vu les caractéristiques et les paramètres qui peuvent impacter le coût. Plusieurs études [DG08, BFG+08, IBY+07, OV11, T+10, BPASP11, ABPH+10, AKB+12, XRZ+13, SAR+14, SJK+13, OKH+13, WC13, KTR12, GARK12, ZCD+12] ont été menées dans ce sens. Kosmann [Kos00] a fait un état de l'art du domaine qui récapitule une bonne partie des techniques de traitement et d'optimisation de requêtes dans les environnements répartis. Ces techniques forment un socle utilisé dans des techniques récentes pour améliorer l'optimisation du traitement ou de l'utilisation de la bande passante. Les travaux de cette décennie ont prêté beaucoup d'attention sur le traitement intensif de gros volumes de données dans les architectures réparties à large échelle. Google développa GFS [GGL03] et BigTable [C+06] pour la gestion de ses applications. Ensuite, il proposa MapReduce [DG08] qui est un paradigme de programmation pour le parallélisme des traitements de gros volumes de données dans de grands clusters de machines en découpant un traitement en un ensemble de tâches qui vont être exécutées parallèlement sur des nœuds différents. Beaucoup d'autres travaux qui ont suivi se sont basés sur les principes de MapReduce. Par ailleurs, des solutions qui diffèrent du principe de MapReduce sont aussi proposées.

Dans la suite de cette section, nous faisons l'état de l'art des solutions d'optimisation qui implémentent ou étendent MapReduce. Puis, nous présentons des solutions d'optimisation de requêtes à large échelle, qui ont des principes différents de ceux de MapReduce. Enfin, nous montrons le positionnement de notre travail dans le contexte de l'optimisation de requêtes.

2.2.1 Optimisation de requêtes avec MapReduce et ses extensions

Dans cette sous-section, nous présentons quelques approches d'optimisation de requêtes qui implémentent ou étendent le paradigme MapReduce, après avoir présenté MapReduce et son fonctionnement.

2.2.1.1 MapReduce

MapReduce est un modèle de partitionnement de traitement (ou tâche) en vue d'une exécution répartie à travers plusieurs machines. Il a été proposé pour simplifier le traitement parallèle à travers un système de calcul réparti [DG08]. Le principe est de décomposer un traitement en un ensemble de traitements plus petits, ou plus précisément de découper un traitement portant sur de très gros volumes de données en traitements identiques portant sur des sous-ensembles de ces données. Les traitements (et leurs données) sont ensuite dispatchées vers différentes machines qui vont les exécuter, puis les résultats sont récupérés et consolidés.

MapReduce se résume en deux phases : le map (phase en amont) qui consiste à la décomposition du traitement en plusieurs (sous-) traitements, et le reduce (phase en aval) qui correspond à la consolidation des résultats issus de la phase map.

La mise en œuvre de ces principes s'appuie sur la manipulation de couples (clé, valeur). Le traitement initial s'effectue sur des couples (clé, valeur), et chacun des traitements intermédiaires également. Chaque traitement intermédiaire renvoie un résultat sous la forme d'un ensemble de couples (clé, valeur), et la fonction reduce combine tous ces résultats en un ensemble de couples (clé, valeur), tel que la clé est unique (un seul couple par clé).

Le paradigme MapReduce est principalement destiné à des traitements de type *batch*, de très grande taille, portant sur de très grands volumes de données. La décomposition typique consiste à découper le volume de données initial en plusieurs volumes plus petits, qui peuvent être traités séparément.

Le paradigme MapReduce est fortement utilisé dans plusieurs applications de traitement de données massives telles que le calcul scientifique et l'analyse de données.

Toutefois, MapReduce se limite à deux fonctions (map et reduce), ce qui nécessite un effort de programmation important : un traitement complexe doit être traduit en un enchaînement de nombreuses tâches. De ce fait, un programme MapReduce est généralement trop spécifique et peu réutilisable.

Beaucoup de travaux ont été menés autour du modèle MapReduce pour son implémentation et/ou son extension. Yahoo développa Hadoop qui est une implémentation en libre accès, de MapReduce. Pour améliorer la flexibilité et les performances, Hadoop a eu plusieurs dérivées.

2.2.1.2 Hadoop et des dérivées

Hadoop implémente le modèle MapReduce. C'est un système conçu pour traiter en batch de très grandes quantités de données tout en tolérant les pannes. Il s'appuie sur le système de fichiers distribué HDFS [SKRC10] pour stocker les données. Il exploite la localité des données en déplaçant les traitements vers les données. Cependant, Hadoop présente des limites de flexibilité et de performances avec une latence minimale de l'ordre de dix secondes et des traitements pouvant atteindre plusieurs heures [XRZ+13]. De ce fait, il n'est pas adapté à des situations où la quantité de données interrogées est petite et aux applications interactives. Pour améliorer la flexibilité et les performances, des extensions de Hadoop ont suivi.
Hive [T+10] est une solution ciblant les application analytiques, basée sur Hadoop et HDFS. Hive offre un accès de plus haut plus niveau que MapReduce en proposant le langage de requêtes HiveQL proche de SQL. Ce qui permet à l'utilisateur d'interroger plus facilement les données. Hive a été conçu par Facebook qui l'utilise dans les traitements de ses gros volumes de données [T+10].

Cependant, il importe de noter que Hive tout comme Hadoop ne présente pas des avantages (en termes de performances) lorsque les données manipulées dans une requête sont de petites tailles (de l'ordre de quelques Mo). Ces solutions basées sur MapReduce, s'avèrent être lentes car une requête est traduite en un enchaînement de tâches map et reduce pour être exécuté, le résultat intermédiaire de chaque tâche est matérialisé dans le système de fichiers (HDFS ou GFS). Ceci engendre un nombre d'entrées/sorties important. Pour bénéficier des avantages des SGBD relationnels en termes d'optimisation locale des traitements, les auteurs de [BPASP11, ABPH+10] ont proposé HadoopDB. Dans ce système, chaque machine héberge un SGBD relationnel (PostgreSql) pour stocker une partie des données et disposer d'un accès efficace aux données. La répartition des données est fixée lors du chargement des données dans les SGBD. HadoopDB utilise Hive pour calculer le plan d'exécution d'une requête et Hadoop pour bénéficier du parallélisme entre les machines et des facilités pour échanger des données entre les machines. Comme Hive, HadoopDB décompose une requête en un enchaînement de tâches map et reduce. Les résultats intermédiaires sont matérialisés dans les SGBD, ce qui a pour inconvénient d'augmenter la latence des requêtes.

2.2.1.3 Spark et Spark SQL

Spark est un système de traitement parallèle de données proposant de stocker les données dans la mémoire principale des machines pour obtenir de meilleures performances. Il peut utiliser le système de fichiers HDFS pour le stockage persistent des données. L'abstraction que Spark utilise pour représenter les données est une collection générique appelée RDD (pour Resilient Distributed Dataset [ZCD+12]). Cette abstraction permet de manipuler des données réparties sur plusieurs machines aussi simplement que des données centralisées. Spark s'appuie sur le paradigme MapReduce pour proposer des opérations algébriques de manipulation de données (sélection, projection, regroupement, etc.) qui transforment des RDD en d'autres RDD. Une RDD peut ainsi être définie comme un enchaînement d'opérations algébriques et peut être recalculée si nécessaire. Cela permet à Spark de garder les données en mémoire en fonction des requêtes et de garantir leur recouvrement en cas de panne. Contrairement à Hadoop et Hive, Spark garde les résultats intermédiaires en mémoire et les déplace sur disque seulement lorsque c'est nécessaire. Le stockage en mémoire permet à Spark d'éviter les congestions dues aux E/S disques notamment pour les résultats intermédiaires d'une tâche map. La notion de RDD permet également à Spark de définir le partitionnement des données au moment de leur création.

Spark SQL [Inc] est une solution récente basée sur les ressources de développement de Shark [XRZ+13]. Shark s'appuie sur Spark pour bénéficier des avantages des RDDs et utilise Hive pour disposer d'une interface de haut niveau pour interroger des données. En outre, il se sert du compilateur de requêtes de Hive pour analyser les requêtes afin de produire un plan logique. Ce plan est

complété par des règles additionnelles fixées par Shark pour générer un plan algébrique composé d'opérations sur des RDDs. Pour optimiser une requête, Shark recueille des statistiques sur les résultats intermédiaires (taille, distribution des valeurs). Cela lui permet de choisir un algorithme efficace pour traiter les opérations algébriques telles de la jointure. Les performances de Shark sont 40 à 100 fois meilleures que celles obtenues avec Hadoop([XRZ+13]). Basé sur Spark, Spark SQL permet de contrôler le partitionnement initial des données. L'optimisation ne considère pas les paramètres dynamiques et hétérogènes des environnements à large échelle notamment la différence de charge entre les sites et les fluctuations de la charge sur un même site au cours du temps.

2.2.1.4 Autres travaux d'optimisation basés sur MapReduce

L'adoption progressive du modèle MapReduce pour le traitement intensif sur de gros volumes de données a suscité beaucoup d'attention. Plusieurs travaux [OKH+13, WC13, KTR12, GARK12] visant à optimiser ces traitements sont menés. Certaines recherches centrent leur optimisation sur la répartition dynamique des données en utilisant les statistiques sur les données et les propriétés des requêtes. D'autres [WC13, OKH+13] visent à factoriser les traitements redondants (par exemple les *scan* du même fichier) en un seul et à partager les résultats des *map* pour les requêtes itératives ou partageant les mêmes traitements sur les mêmes fichiers via une approche de matérialisation de vues. En effet, pour Wang et al., puisque différentes tâches partagent des travaux similaires sur les mêmes données et produisant les mêmes résultats, il y a beaucoup d'opportunités pour optimiser les performances en exploitant le traitement partagé à travers les tâches. Ceci permettra d'éviter les calculs redondants dans un framework MapReduce. Ainsi, ils proposent deux nouvelles techniques d'optimisation multitâches dans le framework MapReduce. La première est une technique de regroupement généralisée de MRShare qui fusionne des tâches multiples en un seul tâche afin que les tâches fusionnés partagent à la fois le *scan* du fichier d'entrée ainsi que la communication du résultat du *map* correspondant. La deuxième est une technique de matérialisation qui permet à plusieurs tâches de partager à la fois le scan du fichier d'entrée ainsi que la communication du résultat du *map* correspondant via une matérialisation partielle des résultats de certains tâches. Avec un objectif proche Onizuka et al. [OKH+13] proposent OptIQ qui est une approche d'optimisation de requêtes itératives sur le framework MapReduce. OptIQ vise à supprimer les traitements redondants pour différentes itérations en utilisant des techniques de matérialisation et d'évaluation incrémentielle de vues.

Ces techniques d'optimisation multi-requêtes (ou de requêtes itératives) analysent les opérations des requêtes et les données impliquées pour trouver les traitements communs (ou partagés). Cependant, elles ne considèrent pas les occupations de calcul des sites qui doivent effectuer les traitements. Or, la charge et les performances du site sont déterminantes pour la latence de la nouvelle tâche.

Toujours dans l'optique d'optimiser MapReduce, des travaux comme [KTR12, GARK12] tentent d'équilibrer la charge à travers les sites avec un mécanisme de répartition dynamique qui pondèrent le contenu d'un site en fonction de ses performances. Ces approches collectent au cours du temps les statistiques sur les données. Les statistiques sur les propriétés des données [KTR12], peuvent être combinées avec un modèle de coût [GARK12] qui dépend des données et des sites redistribuer les données à travers le cluster. Même si ces approches d'équilibrage de la charge intègrent les performances des sites, il importe de constater qu'elles ne prennent pas compte les charges courantes des sites. Or, ce paramètre très fluctuant impacte la latence pour démarrer une tâche.

2.2.2 Quelques solutions spécifiques sur l'optimisation de requêtes

En plus des travaux implémentant ou étendant le modèle MapReduce, plusieurs recherches [SJK+13, AKB+12, SAR+14] ont été menées dans le domaine de l'optimisation de requêtes à large échelle. Nous présentons dans la suite quelques travaux qui ont été menés dans ce sens.

– **Travaux de Schaffner et al (2013) [SJK+13].** considèrent des algorithmes qui contrôlent les ressources (contraction et expansion) d'un cluster de bases de données en mémoire en fonction du comportement des applications au cours du temps tout en maintenant un temps de réponse garanti. Avec le développement des bases de données en mémoire, et l'attractivité des services SaaS du cloud, ils proposent un mécanisme d'optimisation qui garantit des temps de réponse acceptables pour les applications avec un nombre minimal de sites. Cette approche d'optimisation s'appuie sur une technique de placement incrémentielle des données en supposant que la charge est répartie de façon équitable à travers les applications. Cette politique de placement dynamique supporte la réplication et est guidée par la charge due aux applications sur chaque fragment. Connaissant la charge de chaque application sur une donnée et l'ensemble des applications qui exploitent une donnée, les auteurs proposent un algorithme qui détermine le schéma de placement optimal pour garantir un temps de réponse minimal avec le plus petit nombre de sites. Lorsque l'exploitation des données engendre une surcharge sur un site, alors l'algorithme réadapte le schéma de placement pour maintenir une charge acceptable sur chaque site. Cette adaptation dynamique du schéma de placement des données évitant la surcharge d'un site suppose une connaissance parfaite des comportements futurs des applications. Elle peut être efficace, lorsque ce comportement est statique ou régulier et que chaque application utilise les mêmes données en continuité. Cependant, elle peut engendrer un coût de replacement dynamique inutile lorsque le nombre et le comportement des applications sont dynamiques et irréguliers. En effet, RTP se base sur le comportement antérieur des applications pour décider du placement qui serait adapté dans l'avenir. Ceci ne serait pas efficace lorsque de nouvelles applications intègrent le système après le replacement.

– **Travaux de Soliman et al (2014) [SAR+14].** Pour faire face à l'accroissement des données et aux limites des optimiseurs existants (avec des résultats non-optimaux), les auteurs d'Orca, tentent d'améliorer les performances des systèmes en tenant compte dans leur architecture, les nouvelles exigences des applications et leurs développements futurs. Orca est basé sur GreenPlum/Pivotal et est spécialement destiné aux applications analytiques à la demande. Pour ce qui concerne l'optimisation de requêtes, Orca utilise les statistiques sur les propriétés des données. Dans un premier temps, il explore un ensemble de plans en fonction des opérations de la requête. Ensuite, en fonction des informations sur les statistiques des données impliquées, il aménage (par des transformations) les plans d'exécution qui ont été calculés lors de la phase d'exploration. Il effectue une estimation du coût de chaque plan candidat en fonction des statistiques afin de déterminer le plan optimal qui est le plan de moindre coût. Les auteurs de Orca, considère que l'optimisation de la requête est coûteuse en calcul et proposent à cet effet un mécanisme de parallélisme de cette phase.
– **Travaux de Agarwal et al (2012) [AKB+12].** considèrent que la génération du plan d'exécution de la requête avant son exécution n'est pas efficace dans le contexte du large échelle, puisque les propriétés à partir desquelles le plan est calculé sont difficiles à estimer. En effet, pour eux, les propriétés des traitements et des données varient largement au cours du temps. De ce fait, les utiliser de façon fixe pour des estimations a priori mènerait à des performances inefficaces. Ainsi ils proposèrent, RoPE, une approche de ré-optimisation de la requête au cours de son exécution. RoPE collecte les informations sur les traitements et les propriétés des données pendant l'exécution de la requête pour alimenter l'optimiseur de requêtes. Ce dernier utilise les statistiques sur les propriétés des données et des opérations pour adapter les plans d'exécution des requêtes en cours. Les statistiques sont également utilisées pour adapter les plans d'exécution des futures invocations des mêmes tâches avec un optimiseur de requêtes basé sur le coût. Le nouveau plan nécessite un changement global du plan d'exécution de la requête : par exemple réordonner les opérations, choix de l'implémentation de l'opération appropriée et le regroupement des opérations ayant des tâches petites.

2.2.3 Positionnement de notre approche

2.2.3.1 Synthèse

L'état de l'art sur l'optimisation de requêtes à large échelle, nous permet de constater que plusieurs paramètres et mécanismes pouvant améliorer les performances sont pris en compte dans le processus d'optimisation de requêtes. En effet, le placement dynamique en fonction de statistiques [SAR+14] sur les données ainsi que la factorisation de tâches redondantes ou encore la matérialisation de certains résultats peuvent adapter les performances du système en fonction des accès. De la même façon, la ré-optimisation [AKB+12] de requêtes en cours d'exécution peut

Travaux	Placement dynamique	Propriétés des données	Opérations des requêtes	Charges courantes	Temps de réponse borné	Ré-optimisation
Hadoop-Hive	Non	Oui	Oui	Non	Non	Non
Spark SQL	Oui	Oui	Oui	Non	Non	Oui
Travaux de Schaffner et al (2013)	Oui	Oui	Oui	Oui	Non	Non
Travaux de Soliman et al (2014)	Non	Oui	Oui	Non	Non	Oui
Travaux de Agarwal et al. (2012)	Non	Oui	Oui	Non	Non	Non

TABLE 2.3 – Quelques travaux spécifiques sur l'optimisation de requêtes

adapter le plan d'exécution en fonction de paramètres réels du système. En outre, un mécanisme efficace [SJK+13] de stockage des données en mémoire réduit les risques de congestion dus aux accès disque pour les traitements.

Cependant, des paramètres, telle que la disparité et la fluctuation des charges à travers les sites, ne sont pas pris en compte dans ces solutions. En outre, les solutions qui ré-optimisent la requête pendant son traitement, paraissent bénéfiques pour des traitements longs (de l'ordre de plusieurs minutes voire heures), mais peuvent être pénalisantes pour des requêtes de courte durée (moins de dix secondes par exemple) puisque la mise en œuvre de la ré-optimisation engendre un surcoût élevé par rapport au temps pris la requête. Nous récapitulons dans le tableau 2.3 les paramètres d'optimisation qui sont pris en compte par chacune des solutions présentées plus haut. Pour cela, nous montrons la coordination ou non du placement dynamique des données par l'optimisation de requête, l'intégration ou non des propriétés des requêtes, des propriétés des données et les charges courantes des sites. En outre, nous représentons si l'approche d'optimisation vise à borner le temps de réponse de la requête. Enfin, nous montrons si l'approche effectue la ré-optimisation de la requête au cours de son traitement ou non.

2.2.3.2 Positionnement de notre approche

Notre approche d'optimisation considère plusieurs paramètres susceptibles d'affecter les performances du système. En effet, elle guide la répartition dynamique des données dans l'optique

Travaux	Placement dynamique	Propriétés des données	Opérations des requêtes	Charges courantes	Temps de réponse borné	Ré- optimisation
Notre approche	Oui	Oui	Oui	Oui	Oui	Non

TABLE 2.4 – Notre approche d'optimisation de requêtes

de trouver un schéma de placement qui s'adapte davantage aux comportements des applications et qui tient compte de la structure des données et des requêtes. Avec une contrainte de temps de réponse borné, notre approche prend en compte les charges courantes des différents sites (caractérisées par la disparité et fluctuation). En outre, elle utilise les statistiques sur les temps de traitement des fragments sur les sites dans l'estimation du coût de la requête. Notre présentons dans le tableau 2.4 le comportement de notre approche pour les différents paramètres d'optimisation.

2.3 Exploitation des données de biodiversité

Des initiatives visant à améliorer l'exploitation des données de biodiversité ont été menées [Sec, FC, Mos, L., Bio]. La plupart de ces solutions ont des fonctionnalités limitées et préconisent la mise en place de portails dédiés selon une thématique [Mos] ou selon un pays [FC]. Les portails dédiés à des thématiques partagent les données relatives à la thématique et les portails nationaux partagent les données des fournisseurs du même pays. Ce qui diffère de l'objectif initial du GBIF qui est de rendre publiquement accessibles toutes les données de biodiversité à l'échelle mondiale [Hob03, Sec]. BioVeL [Bio] est une solution récente qui offre plus de fonctionnalités pour l'analyse de biodiversité. Elle permet à l'utilisateur de choisir ou charger les sources de données de ses études avant de poser une série de requêtes pour filtrer et préparer ses données. Toutefois, le transfert de sources distantes avant les traitements peut poser des problèmes de disponibilité (latence élevée). De plus, la série de requêtes qui filtre les données pourrait être exprimée par une seule requête complexe afin de réduire la durée totale de l'analyse.

Notre solution quand à elle, place les données aux endroits où elles sont fréquemment utilisées indépendamment de leurs origines et de leurs thèmes. Les utilisateurs peuvent aussi accéder aux données sans se soucier de leur emplacement. Ceci permet de conserver l'objectif initial du GBIF. Notre solution apparait donc comme un complément de ces solutions qui ne partagent pas directement leurs données et/ou leurs ressources de calcul pour répondre à tous les besoins des usagers. Du point de vue des fonctionnalités, des solutions [FC, L.] ont intégré des applications limitées d'analyse de données de biodiversité. La plus avancée de ces solutions est MostiquoMap [Mos] qui est une initiative très intéressante permettant à l'utilisateur de lancer ses analyses via une interface où il choisit des valeurs pour quelques paramètres, puis de

visualiser les résultats sous forme de carte. Cependant, cette application reste très rigide et concerne uniquement les analyses sur les moustiques et donc ne tient pas compte de toutes les analyses possibles sur un ensemble plus large notamment toutes les données de biodiversité à l'échelle mondiale. Notre solution préconise l'assistance des usagers dans l'exploitation qu'ils auront à faire des données primaires de biodiversité. En effet, la solution supporte l'intégration de nouvelles fonctionnalités telles que la possibilité de traiter des requêtes plus complexes correspondant aux nouveaux besoins d'analyse des données de biodiversité.

Chapitre 3

Gestion décentralisée des données de biodiversité

Dans ce chapitre, nous proposons une solution décentralisée pour interroger les données de biodiversité du GBIF. Pour cela, nous présentons d'abord le fonctionnement du système d'information actuel du GBIF. Puis, nous présentons notre solution décentralisée en détaillant ses objectifs, les contraintes de sa faisabilité, les principes généraux de son fonctionnement et son architecture.

3.1 Le système d'information du GBIF

Les chercheurs, étudiants, décisionnaires ont besoins d'accéder aux stocks mondiaux de données sur la biodiversité. La préservation de celle-ci est indispensable pour la survie de l'humanité. Créé en 2001, le Global Biodiversity Information Facility (GBIF) ou Système Mondial d'Information sur la Biodiversité (SMIB) est un consortium international visant à fédérer et partager les données de biodiversité à l'échelle mondiale. Il est reconnu comme étant la référence, pour les données primaires de biodiversité, sur laquelle s'appuient les autres initiatives comme LifeWatch [P.b] et GEOBON [GB]. Le portail du GBIF est un outil fournissant des données et des informations scientifiques fiables et en nombre, sur lesquelles on peut baser des analyses scientifiques valables et des décisions de gestion de la biodiversité (voir les cas d'usage [GBI]).

L'objectif du GBIF est de permettre aux responsables politiques, aux décisionnaires, aux chercheurs et au grand public, partout dans le monde, d'accéder de façon dématérialisée et gratuite aux stocks mondiaux de données primaires sur la biodiversité. Pour atteindre cet objectif, le GBIF travaille en étroite collaboration avec les organisations qui compilent, maintiennent et utilisent les ressources d'information biologiques.

La quantité de données partagées à travers le GBIF est passée de 160 millions d'enregistrements d'observations en 2008 plus à plus de 520 millions en mars 2015. Avec un taux d'accroissement élevé, dû aux partenaires et programmes de mobilisation de plus en plus nombreux, cette quantité atteindrait dans quelques années plusieurs milliards. En outre, la biodiversité, de par sa richesse, englobe plusieurs variétés d'informations du gène à l'écosystème. Le volume important, la variété et la vélocité de la mobilisation des données de biodiversité permettent de qualifier ces données de big data.

3.1.1 Portail du GBIF : architecture et fonctionnement

Le portail du GBIF fournit un accès électronique aux collections mondiales de données primaires sur la biodiversité. Le portail rassemble la description de la plupart des collections de données de biodiversité existant à travers le monde. Pour mettre en œuvre l'intégration des collections (recensement, numérisation, standardisation, ...), le GBIF a tissé un réseau de correspondants nationaux dans de nombreux pays. Le rôle d'un correspondant national est de gérer l'informatisation locale des collections d'un pays.

3.1.1.1 Architecture du portail GBIF

L'architecture du portail du GBIF, représentée sur la figure 3.1, est fondée sur les correspondants nationaux appelés nœud nationaux. Un nœud national dispose de ressources informatiques dédiées à l'intégration des données nationales au sein du portail. Chaque nœud national dispose de fournisseurs de données de biodiversité (data providers) certifiés qui partagent leurs collections à travers le GBIF.

Le portail réalise deux services distincts : premièrement, l'intégration des données et, deuxièmement, l'interrogation des données intégrées.

i) L'architecture pour l'intégration des données dans la base centrale du portail est composée des nœuds nationaux et du nœud central (portail du GBIF). Un protocole tel que DiGIR [P.a], TAPIR [P.c] ou BioCASE, IPT, spécifie l'envoi des données vers le nœud central. Dans ce module, les nœuds nationaux correspondent aux fournisseurs de données primaires de biodiversité.

ii) L'architecture pour l'interrogation des données est actuellement centralisée : toutes les requêtes d'interrogation des usagers sont traitées dans la base de données du nœud central.

Le portail du GBIF dispose d'une base de données qui récapitule toutes les descriptions des occurrences partagées à travers le réseau GBIF. Le contenu de la base suffit pour répondre aux requêtes des usagers.
Les usagers peuvent rechercher les occurrences via un navigateur web par http. Ils peuvent aussi télécharger les données qui sont nécessaires à leurs analyses en utilisant des services web.

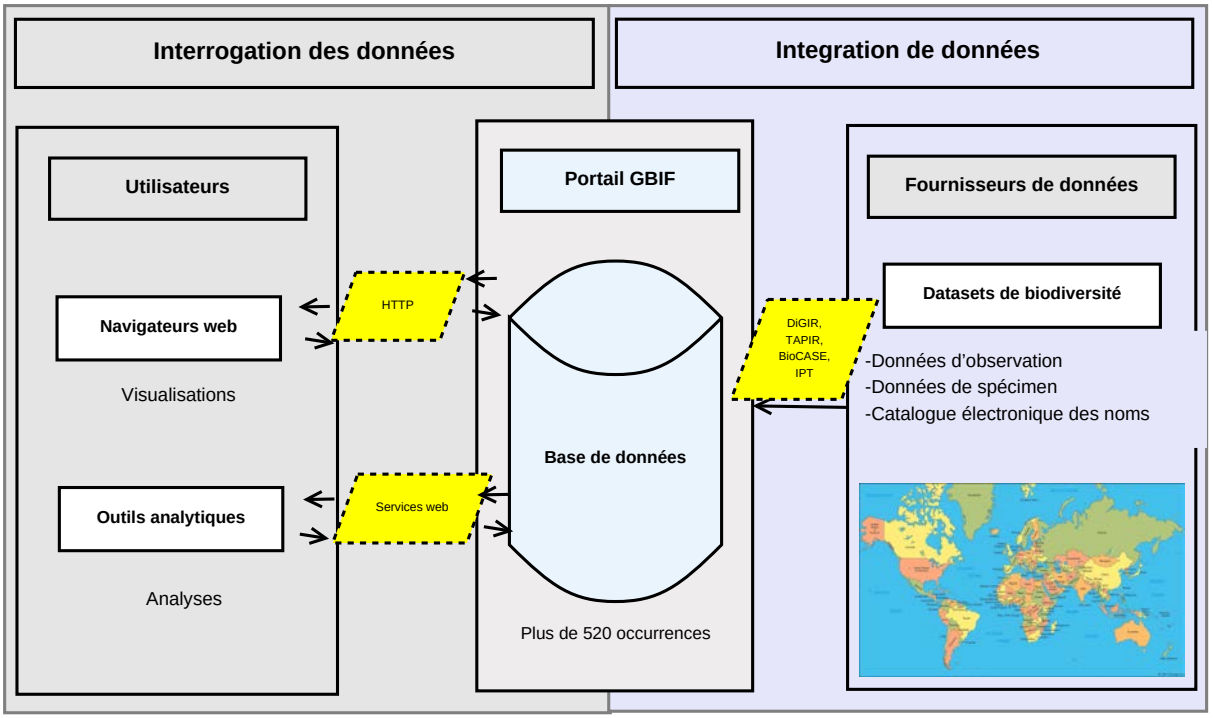

FIGURE 3.1 – Architecture et fonctionnement du GBIF

3.1.1.2 Schéma des données

Nous décrivons le schéma des données du portail du GBIF. La base contient un ensemble de collections. Une collection contient un ensemble d'occurrences. Une occurrence décrit une observation d'un spécimen. Une occurrence contient de nombreux champs taxinomiques ainsi qu'un géo-référencement. Une occurrence constitue l'unité d'information élémentaire de la base de données. Par ailleurs, la provenance des collections est décrite : une collection appartient à un fournisseur, lui-même rattaché à un pays. En termes de taille, la base de données contient plus de 500 millions d'occurrences issues de plus de 15000 collections et 600 fournisseurs. La taille totale de la base est de plusieurs centaines de giga octets. Pour ce qui concerne la structure, la base de données est composée d'une table principale très volumineuse qui contient toutes les informations sur les occurrences de spécimens partagées à travers le réseau, et des tables annexes de petites tailles.

3.1.1.3 Requêtes supportées par le portail

Le portail GBIF permet de rechercher des occurrences d'espèces. Le portail propose deux types de requêtes :

i) une requête par navigation selon un pays, une collection ou une structure taxinomique ;

ii) une requête conjonctive multi critères portant sur un nombre fini de champs (nom scientifique, date, localisation géographique, etc..).

Les requêtes supportées actuellement par le portail, sont des sélections simples sur les occurrences. La forme générale de ces requêtes est la suivante :
*SELECT **
FROM occurrence
WHERE prédicat$_1$ *AND prédicat*$_2$*, AND .....AND prédicat*$_n$

Prédicat$_i$ est de la forme attribut$_i$ = valeur (sauf pour les prédicats portant sur des attributs numériques qui supporte des comparaisons d'inégalité) où attribut$_i$ n'apparait pas dans les autres prédicats. Le résultat d'une requête a une structure fixe : c'est un ensemble d'occurrences avec tous leurs champs. A l'heure actuelle, le portail ne supporte aucun autre type de requêtes ayant des jointures, des projections, des négations, ... La nature simple de ces requêtes limite les possibilités d'interrogation des données disponibles. En effet, pour les cas d'utilisation notés au [Sec], les analyses portent sur un ensemble fini d'espèces dans des dimensions spatiale et temporelle définies. Ce qui oblige les usagers à effectuer des accès multiples au portail pour télécharger les occurrences de chacune des espèces nécessaires à l'analyse et les garder en local. Cela est coûteux en termes d'accès, de transferts et de stockage surtout lorsque le volume de données à manipuler est important. Une fois les données recopiées localement, l'utilisateur effectue des calculs de regroupement (densité, abondance, distribution, découpage temporel, découpage spatial, ...) sur les occurrences d'espèces. Par ailleurs, le portail propose deux types d'interfaces pour l'interrogation : une interface de navigation par http et une interface programmatique, permettant d'automatiser l'interrogation sous la forme d'un service web. Le service web peut être ainsi utilisé directement par de nombreuses applications tierces qui étudient divers aspects de la biodiversité.

La facilité d'accès automatisé accroit fortement le nombre de requêtes que doit traiter le portail, ce qui accentue le besoin pour une solution de traitement de requêtes plus efficace.

3.1.2 Les données de biodiversité du GBIF

3.1.2.1 Type de données

On note deux grandes catégories de données qui sont manipulées à travers le réseau GBIF. Ce sont les données primaires de biodiversité et les méta-données et peuvent être formées de sous catégories.

– *Données primaires de biodiversité*

Le GBIF s'est très tôt focalisé dans la numérisation des données et la mobilisation de ses activités autour des spécimens. Un enregistrement ou occurrence de donnée primaires de biodiversité décrit une observation de spécimen. Les données primaires sont des informations de descriptions taxonomiques, spatiales et temporelles, ... des observations. Ces données combinées avec les données environnementales (élévation, température, précipitation, type de sol, couverture forestière, etc.) aident à :

- *Prédire* la distribution d'espèces selon divers scénarios de changements climatiques, écologiques, etc.
- *Prévoir* des sites pour la protection de la biodiversité, etc.
- *Anticiper* les régions nécessitant des actions précises (combattre les espèces invasives, mettre en place des mesures sanitaires, etc.).

Le GBIF a initié des activités encourageant la découverte et la mobilisation d'autres types de données de biodiversité telles que les ressources multimédia associées à une observation de spécimen.

- *Les méta-données*

  Les méta-données sont « les données décrivant les données ». Selon [Saa05], elles fournissent des informations sur les aspects « qui, quoi, où et quand » des données. Pour les fournisseurs, les méta-données sont utilisées pour documenter les données en vue d'informer les utilisateurs de leurs caractéristiques et de leurs localisations. Quant aux consommateurs, les méta-données sont utilisées pour la découverte de données et la détermination des opportunités offertes en fonctions des besoins particuliers.

3.1.2.2 Structure hiérarchique des données de biodiversité

Les données de biodiversité ont la particularité d'avoir une structure hiérarchique selon plusieurs dimensions. En effet les dimensions de la classification taxinomique, géographique, temporelle et fournisseurs présentent chacune une structure hiérarchique. Dans notre travail, nous nous limitons aux dimensions taxinomique et géographique qui sont les plus fréquemment utilisées dans les analyses de données notamment en modélisation de niche écologique.

- *Structure hiérarchique de la taxinomie.* La classification taxinomique des espèces en biodiversité présente une structure hiérarchique dans laquelle un ensemble d'occurrence compose une espèce (ou sous-espèce) qui, avec d'autres espèces, forment un genre appartenant à une famille. Ce processus se poursuit jusqu'au règne.
- *Structure hiérarchique de la géographie.* Les données géographiques présentent une organisation hiérarchique où chaque point d'observation fait partie d'une zone qui appartient à une région d'un pays. Un ensemble de pays forme un continent. L'ensemble des continents forme le globe.

3.1.3 Cas d'utilisation des données de biodiversité

Dans cette section, nous présentons les cas d'utilisation des données de biodiversité du GBIF. Nous développons le cas de la modélisation de niche écologique et présentons quelques cas [GBI] recensés sur le portail GBIF.

3.1.4 Modélisation de niche écologique

3.1.4.1 Niche écologique d'une espèce

La niche écologique d'une espèce est l'ensemble des paramètres et conditions qui permettent à l'espèce d'accomplir son cycle biologique et sa reproduction à l'intérieur d'une plage limitée de variations environnementales d'origine abiotique (facteurs physiques et chimiques) et biotique (interaction entre espèces) [SB09]. Hutchinson a précisé ce concept en définissant la niche écologique d'une espèce comme un hyper-volume multidimensionnel dans lequel chaque dimension représente un paramètre biotique ou abiotique conditionnant la présence de l'espèce. La niche fondamentale est l'ensemble des ressources potentielles qu'une espèce peut utiliser dans son milieu lorsque les conditions sont idéales (absence de compétition ou prédation). La niche réalisée (ou niche réelle) est l'ensemble des ressources réellement utilisées par une espèce en conditions naturelles (compétition pour les ressources, contrôle pour un consommateur, prédateur ou herbivore, contrôle par un parasite, absence d'une autre espèce comme par exemple un pollinisateur).

3.1.4.2 Modélisation de niche écologique

La modélisation de la niche écologique d'une espèce est assez complexe et est limitée par la méconnaissance de certaines interactions entre l'espèce à modéliser et d'autres espèces qu'on ignore et aussi par le manque d'information sur le comportement biologique de l'espèce. La modélisation statistique de la niche écologique d'une espèce consiste à construire une fonction de paramètres environnementaux qui prédit la probabilité de présence de l'espèce. Elle est faite à partir d'un jeu de données de calibration comprenant des données de présence/absence ou d'abondance de l'espèce et des valeurs de paramètres environnementaux aux sites d'observation [SB09]. L'objectif de la modélisation de la niche écologique d'une espèce est d'étudier (comprendre, trouver, prédire) les comportements (distribution, migration, menace d'extinction, possibilité d'établissement, ...) de l'espèce vis-à-vis des facteurs environnementaux selon les dimensions spatio-temporelles de l'étude. On trouve des applications de la modélisation de niche écologique en agronomie (impact des abeilles sur une fleur, orientation vers une nouvelle espèce

d'arachide, ...), en écologie (dégager les zones de protection pour une espèce menacée, migration des hirondelles en hiver, ...), en élevage (rendement d'une espèce herbivores en fonction des ressources et cohabitation entre plusieurs espèces), ...

3.1.4.3 Données pour modéliser la niche écologique d'une espèce

Pour modéliser la niche écologique d'une espèce, il est nécessaire de disposer d'un jeu de données de calibration associant des données d'occurrence de l'espèce à des valeurs de paramètres environnementaux sur la zone de l'étude. Les données d'occurrence de l'espèce peuvent être des données de présence/absence ou des données d'abondance. Les paramètres environnementaux peuvent être des données physico-chimiques (températures, salinité, rayonnement solaire) qui caractérisent la zone d'étude et des données d'interaction avec d'autres espèces (co-occurrence avec d'autres espèces).

3.1.5 Autres cas d'utilisation des données de biodiversité

En plus de la modélisation de niche écologique, plusieurs autres cas d'usage des données de biodiversité sont recensés au [GBI]. Ces études consistent à analyser les réseaux d'interaction entre espèces, l'évolution des espèces, les risques d'extinction, l'importance socio-économique, les impacts des changements climatiques, etc. Parmi ces cas, on peut citer l'étude pour cartographier les animaux hôtes du virus Ebola et l'étude de l'importance des espèces rares sur l'écosystème.

Comme la modélisation de la niche écologique, ces cas d'usage nécessitent des données d'occurrences (observation) des taxons qu'ils étudient dans la zone d'étude. Généralement des traitements supplémentaires de regroupement (densité, abondance) sont effectués sur les occurrences. Les résultats de ces traitements sur les observations sont complétés par des données physico-chimiques de l'environnement (température, précipitation, ...) et des connaissances sémantiques sur les taxons.

3.1.6 Utilisation des données du GBIF pour l'analyse de biodiversité

L'analyse de biodiversité nécessite des données de présence/absence ou d'abondance des taxons étudiés et des données environnementales dans la zone l'étude. Le GBIF fournit des données de présence (observation) et devrait permettre de calculer l'abondance de l'espèce dans les zones d'étude grâce au géo-référencement des données. Pour ce qui concerne les paramètres environnementaux, le GBIF peut fournir des données biotiques (interaction avec d'autres espèces) grâce à des requêtes de co-occurrences et les données abiotiques sont obtenues auprès d'autres sources comme bioclim, worldclim [Wor]. L'article [GPA+13] illustre un cas d'utilisation des données du

GBIF complétées avec d'autres données pour l'étude des relations entre les abeilles et les plantes. Dans cette étude, l'utilisateur calcule dans les densités de co-occurrence entre abeilles et plantes en fonction d'un maillage. La cohabitation est considérée pour les zones où les densités sont supérieures à une valeur minimale. Pour comprendre ou modéliser une espèce, il est nécessaire de disposer de connaissances basiques sur la densité ou l'abondance de l'espèce dans chacune des différentes zones de l'étude. Ces informations d'entrée aux analyses nécessitent des traitements d'agrégation et de jointure sur les espèces et les dimensions spatiale et/ou temporelle, que le GBIF n'effectue pas en amont actuellement.

3.2 Interrogation décentralisée des données du GBIF

Pour répondre aux questions de recherche qui visent à optimiser l'interrogation des données de biodiversité, nous proposons une solution décentralisée pour répartir les traitements des requêtes et les données interrogées à travers plusieurs machines. Cela permet d'allouer plus de ressources de stockage et de calcul aux applications. L'objectif est de satisfaire toutes les demandes croissantes des utilisateurs tout en évitant les congestions (passage à l'échelle difficile) qui serait dues à la concentration des requêtes sur un service de requêtes centralisé. La réalisation de la solution nécessite une méthode de résolution qui doit tenir compte des contraintes imposées par le fonctionnement actuel du portail GBIF. La figure 3.2 montre l'architecture générale de notre solution décentralisée.

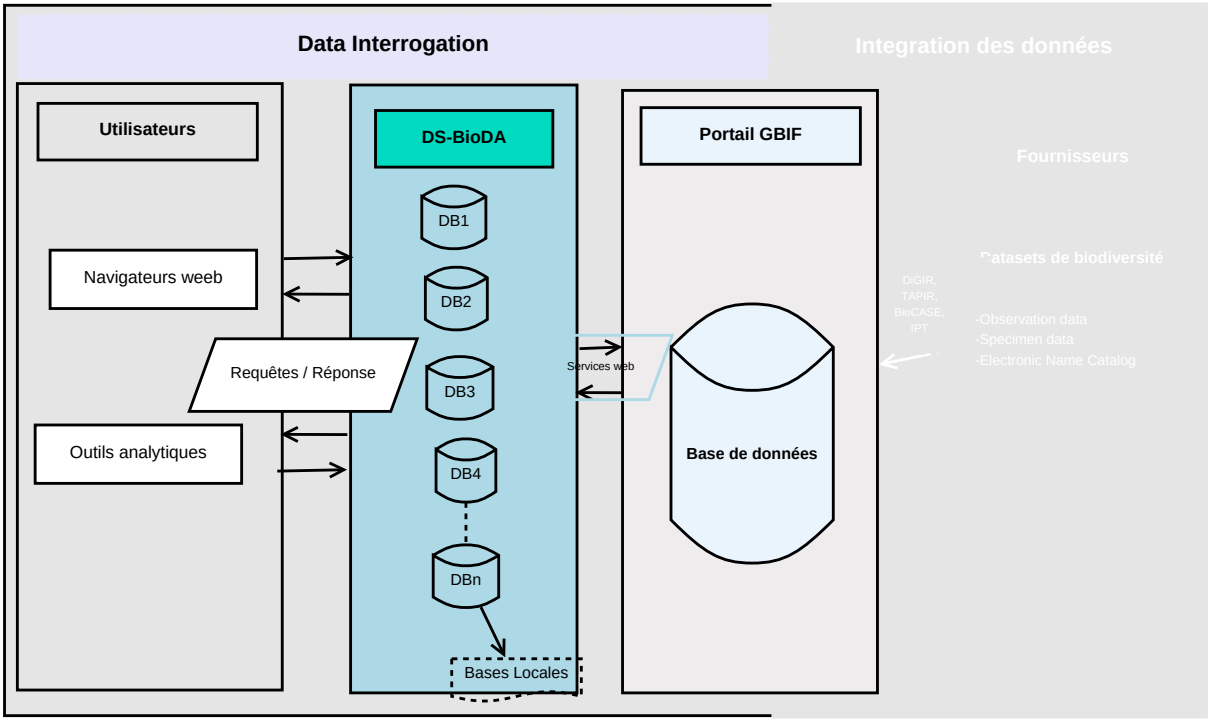

FIGURE 3.2 – Architecture décentralisée pour interroger les données de biodiversité

Après avoir défini les objectifs de la solution décentralisée, nous présentons les principes de fonctionnement qui permettent de garantir sa réactivité vis-à-vis des usagers et le passage à l'échelle des applications. Enfin, nous présentons les composants de l'architecture décentralisée et leurs interactions dans le fonctionnement du système.

3.2.1 Objectifs

Nous détaillons les objectifs que nous avons énoncés à la section 1.1.2La solution doit :

i) étendre les fonctionnalités d'interrogation des données pour pouvoir traiter les requêtes complexes des utilisateurs afin de favoriser la croissance des traitements de données de biodiversité. En plus des requêtes de sélection avec des prédicats simples qui sont disponibles actuellement sur le portail, la solution doit permettre aux utilisateurs de poser facilement des requêtes complexes et interactives pour leurs analyses. Avec un langage de requêtes de haut niveau comme SQL, l'utilisateur doit pouvoir poser des requêtes avec des opérations de jointure, de regroupement, etc. sans limitations sur les portions de données qui l'intéressent. Il s'agit de passer de la sélection simple d'occurrences aux analyses de biodiversité. Les données interrogées par un utilisateur pour une analyse pourraient être décomposées en des portions disjointes. De la même façon, certaines opérations d'une requête pourraient être exécutées indépendamment sur les portions disjointes des données impliquées.

ii) garantir la disponibilité du service d'interrogation des données en utilisant toutes les ressources matérielles et techniques existantes pour répondre aux requêtes dans des délais impartis (temps de réponse borné). Pour cela, la solution doit décharger le GBIF des tâches d'interrogation et doit supporter le parallélisme inter-requêtes et le parallélisme intra-requête. En effet, la solution doit supporter le traitement simultané de plusieurs requêtes indépendantes portant des données disjointes. En outre, elle doit supporter le traitement réparti d'une requête pour bénéficier du parallélisme lorsque les données nécessaires sont réparties à travers plusieurs machines. De plus, la solution doit garantir la disponibilité d'une donnée lorsque celle-ci est sollicitée. La garantie de la disponibilité est d'autant plus cruciale que le support de nouvelles fonctionnalités augmenterait davantage le nombre d'utilisateurs qui vont générer des quantités très importantes de traitement. De ce fait, la solution doit aussi garantir le passage à l'échelle des données et des traitements.

3.2.2 Contraintes

Pour atteindre les objectifs de la solution, nous détaillons certaines contraintes qui sont déjà énoncées à la section 1.1.3.La solution décentralisée pour l'interrogation des données du GBIF doit :

i) se coupler avec le portail existant, sans nécessiter de modifier le portail actuel. Ceci est impératif pour que la solution soit utilisable en réalité. Cela impose de compléter le service de requêtes existant avec d'autres services de requêtes complexes venant relayer (de manière transparente pour les utilisateurs) le portail centralisé. Le nouveau service de requêtes est alors chargé de satisfaire les interrogations des utilisateurs. Cette contrainte rend la solution non-intrusive par rapport au fonctionnement actuel du GBIF.

ii) séparer entièrement l'étape de traitement des requêtes et l'étape d'intégration de nouvelles données. Les requêtes sont par conséquent évaluées sur un cliché (snapshot) des données. La mise à jour des données issues de l'étape d'intégration est effectuée en dehors des périodes de requêtes. Dans le contexte de l'analyse de données de biodiversité portant sur plusieurs années, les utilisateurs tolèrent l'absence temporaire des données les plus récentes dans le cliché qu'ils interrogent. Cette contrainte garantit un fonctionnement autonome de la solution pour l'interrogation des données du GBIF. Les contraintes (i) et (ii) garantissent un service de requêtes complexes indépendant du GBIF.

iii) adapter la stratégie de répartition des données aux capacités des ressources décentralisées. Avec des ressources limitées, nombreuses, proches des utilisateurs et faiblement connectées, la solution doit garantir la disponibilité des données demandées par les utilisateurs. On considère que la capacité de stockage disponible dans un site local ne peut contenir toute la base de données du GBIF. De ce fait une réplication totale de la base complète du GBIF vers les sites locaux est à exclure.

3.2.3 Principes généraux de fonctionnement

Les traitements analytiques qui correspondent aux interrogations des données sont plus fréquents que les modifications dans le système du GBIF. Avec l'accroissement des usagers et le support de requêtes complexes, ces traitements deviennent de plus en plus gourmands en ressources.

Un exemple de requête analytique consiste à étudier les comportements d'un ensemble d'espèces dans une zone géographique. Les étapes du traitement de cette requête sont, d'abord, de sélectionner les espèces à analyser dans la zone de l'étude, ensuite, de faire des regroupements des occurrences de chacune des espèces et, enfin, les données de calibrage souvent environnementales permettent de cerner l'étude afin de tirer les conclusions sur les comportements de chaque espèce.

Pour atteindre les objectifs de la solution décentralisée en tenant compte de la nature des traitements, nous centrons notre solution autour de trois axes que sont le besoin de temps de réponse borné, un mécanisme de répartition dynamique de données et une stratégie de traitement et d'optimisation dynamique de requêtes dans un environnement réparti hétérogène.

3.2.3.1 Temps de réponse borné

Le temps de réponse d'une requête doit être limité pour satisfaire l'utilisateur. En outre, le temps de réponse borné joue un rôle important pour garantir la disponibilité du service d'interrogation des données et supporter des requêtes complexes et interactives. Le temps de réponse d'une requête dépend de plusieurs paramètres : charges et performances des sites, localisations des données impliquées, coûts de transfert, nature des opérations. Une stratégie d'optimisation est appliquée à ces paramètres pour déterminer le plan d'exécution de la requête en fonction de la répartition des données.

3.2.3.2 Répartition dynamique de données

Pour garantir la disponibilité des données interrogées, nous proposons des protocoles appropriées pour coordonner les décisions de fragmentation, de placement et de réplication ainsi que l'accès au portail GBIF. Ces protocoles, basés sur les spécificités des analyses en biodiversité, exploitent l'organisation hiérarchique des données de biodiversité. Une stratégie de répartition dynamique à la demande est utilisée pour la répartition des données à travers les sites locaux. Toujours dans l'objectif de garantir la disponibilité des données et de borner les temps de réponse, des réplications de données entre les sites locaux peuvent être réalisées. Une approche d'optimisation contrôle dynamiquement les décisions de réplication des données.

3.2.3.3 Répartition dynamique des traitements

Les requêtes d'interrogation peuvent nécessiter des ressources de calcul qui dépassent les capacités d'une seule machine, ou bien une machine peut être trop chargée pour garantir un temps de réponse borné. En outre, les données impliquées par une requête peuvent être réparties à travers plusieurs sites. Dans ces cas, la répartition du traitement de la requête à travers plusieurs sites peut être avantageuse pour atteindre l'objectif de garantir un temps de réponse borné en parallélisant certaines opérations et en évitant les transferts de données inutiles. Pour cela, nous mettons en œuvre des protocoles de coordination du traitement réparti de la requête et une approche d'optimisation qui calcule le plan d'exécution de la requête.

Nous développerons ces principes à travers les propositions suivantes qui sont détaillées plus loin :

- Une architecture décentralisée composée d'un ensemble de machines qui hébergent chacune une base de données locale et capable de traiter des requêtes d'utilisateur.
- Une stratégie de répartition dynamique des données à la demande à travers les sites en fonction des prédicats des requêtes des utilisateurs et des états (contenus, charges et performances) des sites.

– Un mécanisme de traitement de requêtes de biodiversité adapté au contexte de la solution décentralisée.
– Une approche d'optimisation basée sur le coût d'une requête qui permet de garantir un temps de réponse borné pour chaque requête et le passage à l'échelle des données et des applications.

3.2.4 Architecture et composants

Nous proposons une architecture répartie qui s'interpose entre les utilisateurs et le portail GBIF. L'objectif de cette architecture est de décharger le portail des tâches d'interrogation et d'offrir aux usagers la possibilité de traiter leurs demandes sans limitation et dans des délais raisonnables. La figure 3.3 montre l'architecture proposée avec les principaux composants et leurs interactions.

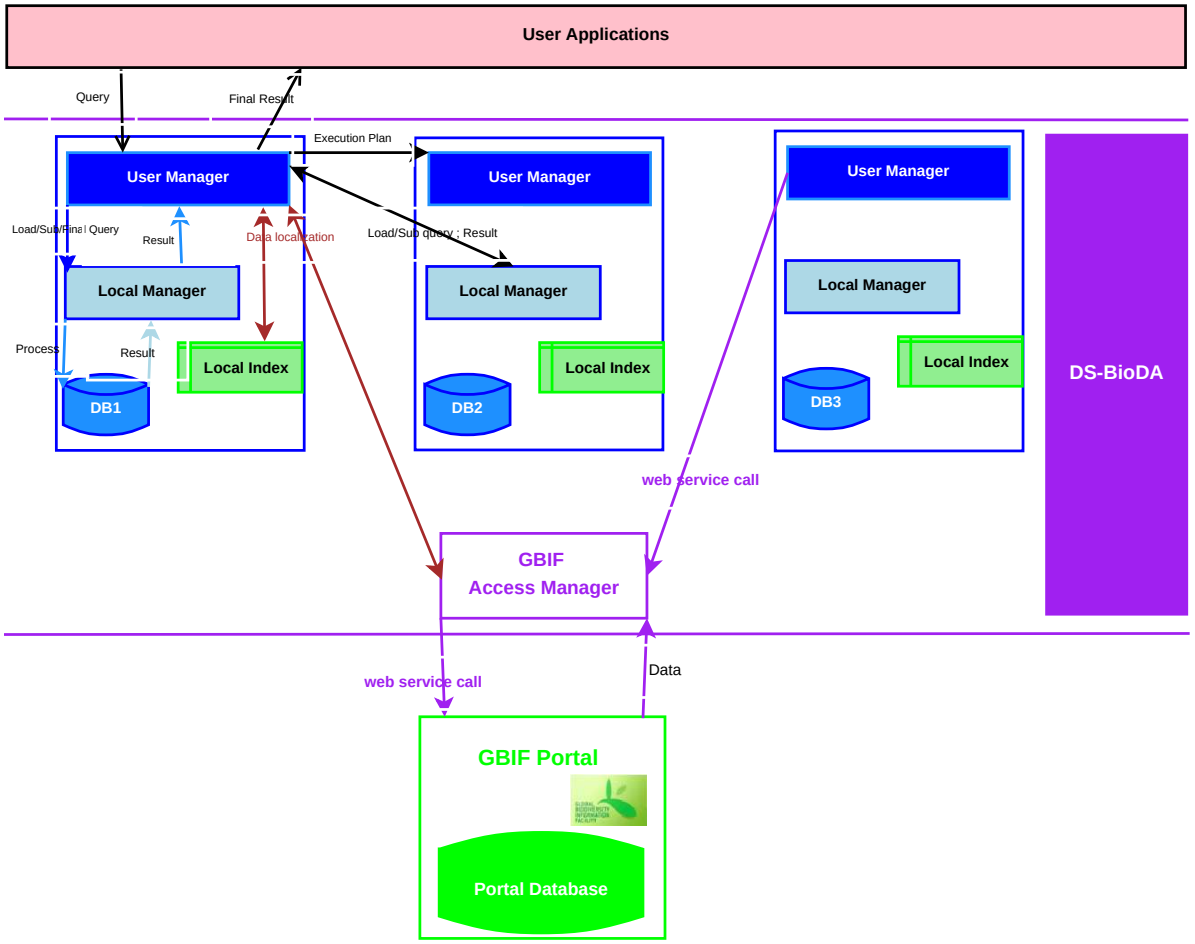

FIGURE 3.3 – Architecture modulaire pour le traitement décentralisé de requêtes

– *Le portail du GBIF :* c'est un entrepôt récapitulant l'ensemble des données primaires de biodiversité accessibles à travers le réseau GBIF [Hob03, Sec]. Nous accédons aux données de la base du portail par des appels web service.

– *Les utilisateurs :* (user applications) sont les usagers qui interrogent les données de biodiversité du portail. Un utilisateur se connecte au site le plus proche via l'interface du gestionnaire des requêtes d'utilisateurs (User Manager), pour poser ses requêtes.
– *Le Gestionnaire des accès GBIF* (GAM pour GBIF Access Manager), représenté à la figure 3.4, coordonne les accès au GBIF et garde les informations sur l'état courant du système concernant les contenus et les charges des sites. Il dispose d'un catalogue global du système et un gestionnaire des appels web service du GBIF.

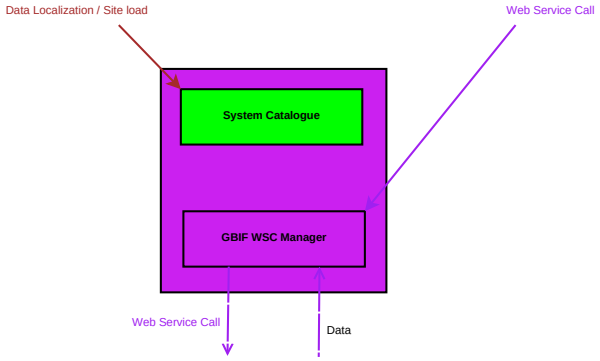

FIGURE 3.4 – Composants du gestionnaire des accès au portail GBIF

  – *Le catalogue du système* (System Catalogue) contient les informations de localisation de tous les fragments et leurs répliques disponibles à travers le système décentralisé et dispose des informations sur l'état de chaque site notamment sa charge courante. Il est utilisé par les gestionnaires de requêtes pour compléter les localisations des fragments lors de l'analyse de la requête.
  – *Le gestionnaire des appels de service web* (GBIF WSC Manager) coordonne les accès au portail GBIF pour télécharger des fragments sollicités par les requêtes et manquants du système décentralisé. En outre, il permet d'éviter les accès redondants au GBIF pour lire la même donnée ou un fragment qui dispose déjà de répliques dans les sites locaux. Il est chargé, de recevoir toutes les demandes d'accès au GBIF, de formuler les appels web service correspondants qu'il soumet au portail GBIF. Il transmet les résultats du GBIF aux sites destinataires. Il est aussi chargé de la gestion de mises à jour des fragments locaux par rapport à ceux du portail GBIF.
– *Sites locaux :* Les sites locaux disposent chacun d'une base de données locale qui héberge les fragments de données sollicités dans les traitements. Ils sont reliés par un réseau et collaborent pour échanger des données et traiter les requêtes. En effet, les fragments sollicités par un utilisateur sont stockés dans les bases de données locales et peuvent être réutilisés par d'autres utilisateurs. Ce qui permet de réduire davantage les traitements d'interrogation au portail GBIF. Chaque site local, S_i, dispose des modules suivants.

- *Une base de données locale (DB_i)* qui est vide au démarrage et est alimentée au fur et à mesure que des traitements lui sont soumis. En effet, pour chaque traitement, les données associées sont placées dans la base du site désigné pour l'effectuer.
- *Un index local :* (Local Index) chaque site S_i dispose d'un catalogue local qui indexe chaque fragment qu'il héberge. Ce catalogue est complété par les informations concernant les localisations de fragments distants des requêtes d'utilisateur soumises au site S_i. Composant essentiel, le catalogue, grâce à son contenu de localisation des fragments, est utilisé par le User Manager lors du processus d'analyse et d'optimisation de la requête pour calculer le plan d'exécution.
- *Un gestionnaire de requêtes* (User Manager), fournit une interface conviviale de haut niveau aux usagers où ils pourront soumettre leurs demandes sans limitation. Il effectue les tâches d'analyse, d'optimisation et de coordination des requêtes d'utilisateurs. Pour cela, il utilise les indexes local et global et l'algorithme d'optimisation pour le calcul du plan d'exécution d'une requête.
- *Un gestionnaire local* (Local Manager), est chargé de la gestion des traitements de toutes les requêtes et les sous-requêtes au niveau de la base de données locale. Il interagit avec les User Manager local ou distants pour recevoir des requêtes ou retourner des résultats. Il dispose d'une file d'attente qui contient les requêtes en attente lorsque le SGBD local est occupé. Il journalise aussi les temps d'exécution de chaque traitement qui est effectué avec la base de données locale. De ce fait, il connait la charge courante des traitements en attente sur le site local.

3.3 Conclusion

Dans ce chapitre, nous avons présenté le système d'information actuel du GBIF. Puis nous avons proposé une solution décentralisée pour décharger le portail du GBIF des tâches d'interrogation des données de biodiversité. Nous avons montré les objectifs de la solution et les contraintes de sa faisabilité et avons défini les principes de fonctionnement qui permettent de garantir la réactivité du système avec le passage à l'échelle des données et des traitements. Nous avons également présenté l'architecture générale et l'architecture détaillée de la solution décentralisée, en montrant ses différents composants et leurs interactions. Dans les chapitres qui suivent, nous présentons la stratégie de répartition des données interrogées à travers les sites locaux et les mécanismes de traitement et d'optimisation des requêtes à travers cette architecture.

Chapitre 4

Répartition dynamique des données à la demande

L'organisation des données dans un environnement réparti joue un rôle essentiel dans les performances du système. Elle est définie par une stratégie de répartition des données à travers les sites. Cette politique s'effectue généralement en trois phases successives que sont : la fragmentation des données, l'allocation des partitions (ou fragments) à travers les sites et éventuellement la réplication de fragments. L'application de la stratégie de répartition des données aboutit à un schéma de placement des données. Ce schéma de placement des données est dit dynamique lorsqu'au moins une phase de la stratégie de répartition est dynamique.

L'objectif de ce chapitre est de présenter les différentes phases de notre approche de répartition dynamique des données de biodiversité à travers l'architecture décentralisée présentée à la section 3.2. Notre approche de répartition des données est basée la structure hiérarchique des données de biodiversité et les spécificités des requêtes d'analyse qui les exploitent (4.1 4.2).

Nous complétons notre politique de répartition dynamique des données avec :

i) une stratégie de remplacement qui a pour objectif d'optimiser les ressources de stockage tout en garantissant la disponibilité des données (4.3).

ii) un mécanisme d'indexation et de localisation rapide des fragments à travers les sites(4.4).

iii) une politique de gestion des mises à jour des fragments et de leurs répliques(4.5).

4.1 Fragmentation multidimensionnelle des données à la demande

Les données et les requêtes de biodiversité ont des spécificités qui peuvent être exploitées pour la fragmentation. Dans la suite de cette section, nous présentons la structure des données de biodiversité et les spécificités des requêtes avant de montrer notre stratégie de fragmentation des données dans notre système.

4.1.1 Organisation hiérarchique des données de biodiversité

Comme nous l'avons mentionné dans la présentation du GBIF, les données de biodiversité ont la particularité d'avoir une structure hiérarchique selon plusieurs dimensions. En effet, les dimensions de la classification taxinomique, géographique et temporelle, qui sont utilisées dans la description d'une observation (occurrence), présentent chacune une structure hiérarchique. En outre, ces dimensions sont généralement utilisées dans les analyses de données de biodiversité notamment en modélisation de niche écologique. Les dimensions taxinomique et géo-spatiale sont les plus fréquentes dans les requêtes d'analyse.

4.1.1.1 Classification taxinomique

La taxinomie (ou taxonomie) permet de décrire les organismes vivants et de les regrouper en entités appelées taxons. Elle permet d'identifier, de nommer et de classer les taxons et de les reconnaitre via des clés de détermination dichotomiques. La taxonomie fixe les critères et les règles de classification des taxons.

La classification taxinomique propose une hiérarchie codifiée. Elle fixe 7 rangs principaux (en gras) et 5 rangs secondaires, présentée, dans l'ordre décroissant, de la façon suivante :
Monde vivant : **règne** $\longrightarrow$ **embranchement, division ou phylum** $\longrightarrow$ **classe** $\longrightarrow$ **ordre** $\longrightarrow$ **famille** $\longrightarrow$ tribu $\longrightarrow$ **genre** $\longrightarrow$ section $\longrightarrow$ série $\longrightarrow$ **espèce** $\longrightarrow$ variété $\longrightarrow$ forme.

La figure 4.1 montre la hiérarchie de la classification taxinomique avec une illustration de la taxonomie de l'espèce humaine.

La classification hiérarchique taxonomique est une structure arborescente où chaque nœud correspond à un taxon. Pour accéder à un nœud, on part de la racine (monde vivant) pour suivre les descendants successifs qui mènent au nœud recherché. La classification taxonomique est standardisée à travers le programme *Catalogue of Life*. Celui-ci consiste en un catalogue qui normalise et donne les informations de relations sémantiques des taxons du monde vivant.

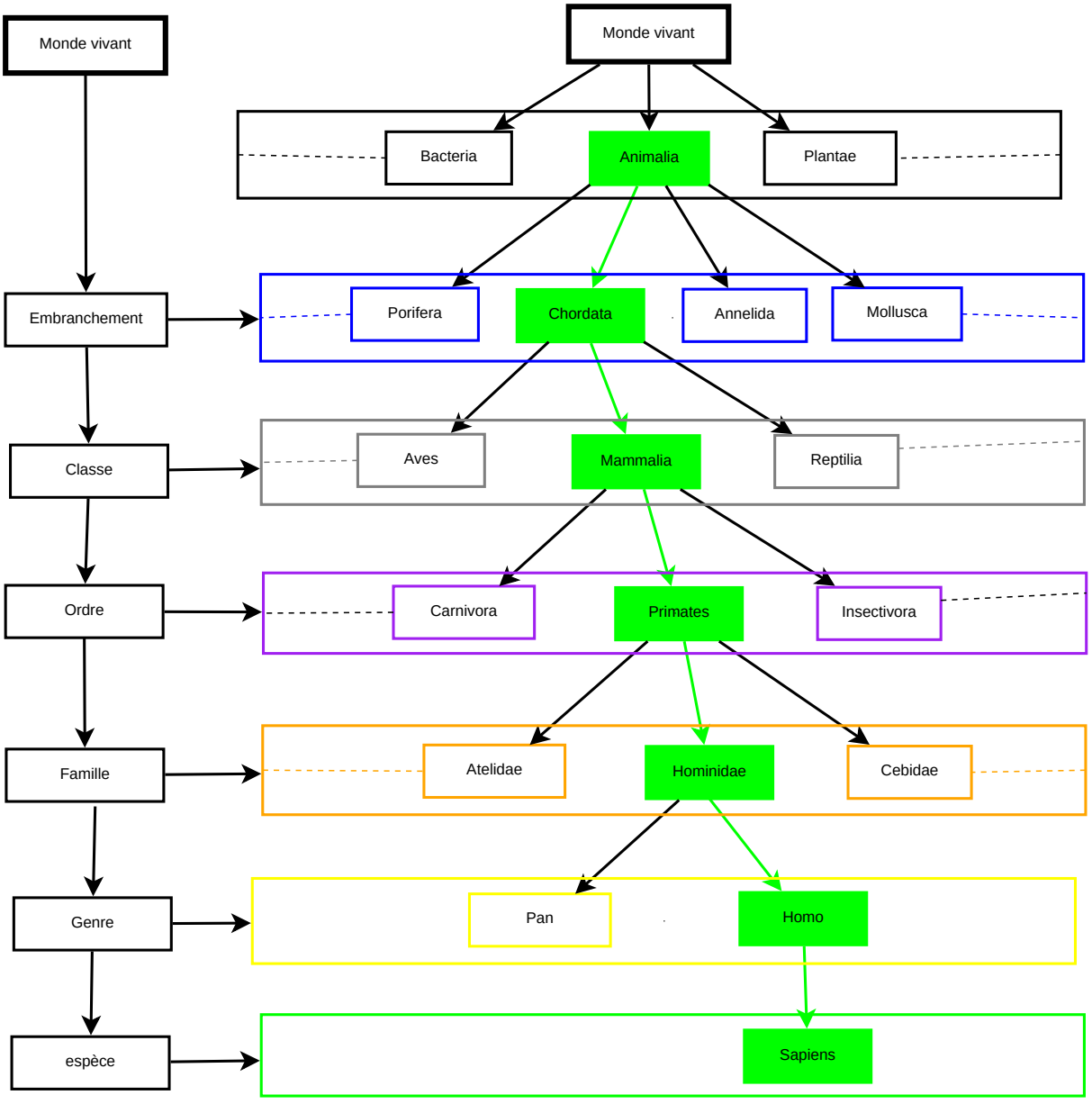

FIGURE 4.1 – Structure hiérarchique de la classification taxinomique

Une observation ou occurrence consiste à une instanciation d'une espèce (niveau le plus bas de la hiérarchie). La description taxonomique d'une occurrence, consiste à renseigner toutes ses propriétés taxonomiques. Ces dernières correspondent à son nom et ceux de ses antécédents.

Definition 4.1. Un nœud taxonomique est défini par la concaténation de l'identifiant de son parent direct (supérieur hiérarchique) et son identifiant dans son groupe (éléments issus directement du même parent).

4.1.1.2 Structure géo-spatiale

Partant du constat que le monde entier peut être considéré comme une zone géographique qui peut être divisée en plusieurs cellules (ou sous-zones) disjointes, on conclut que la description géo-spatiale présente une organisation hiérarchique que nous illustrons à travers la figure 4.2. Cette figure montre un point d'observation qui se trouve dans une cellule (ou pavé ou maille) en jaune. Cette dernière se trouve dans un rectangle plus grand en bleu qui contient d'autres cellules. Ce rectangle est aussi contenu dans un autre rectangle violet plus grand ayant d'autres rectangles. Cette même structure se poursuit jusqu'au rectangle global en orange qui correspond à une représentation plane du monde. Cet exemple nous permet de voir une hiérarchie entre un rectangle et les mailles (ou rectangles plus petits) qu'il contient.

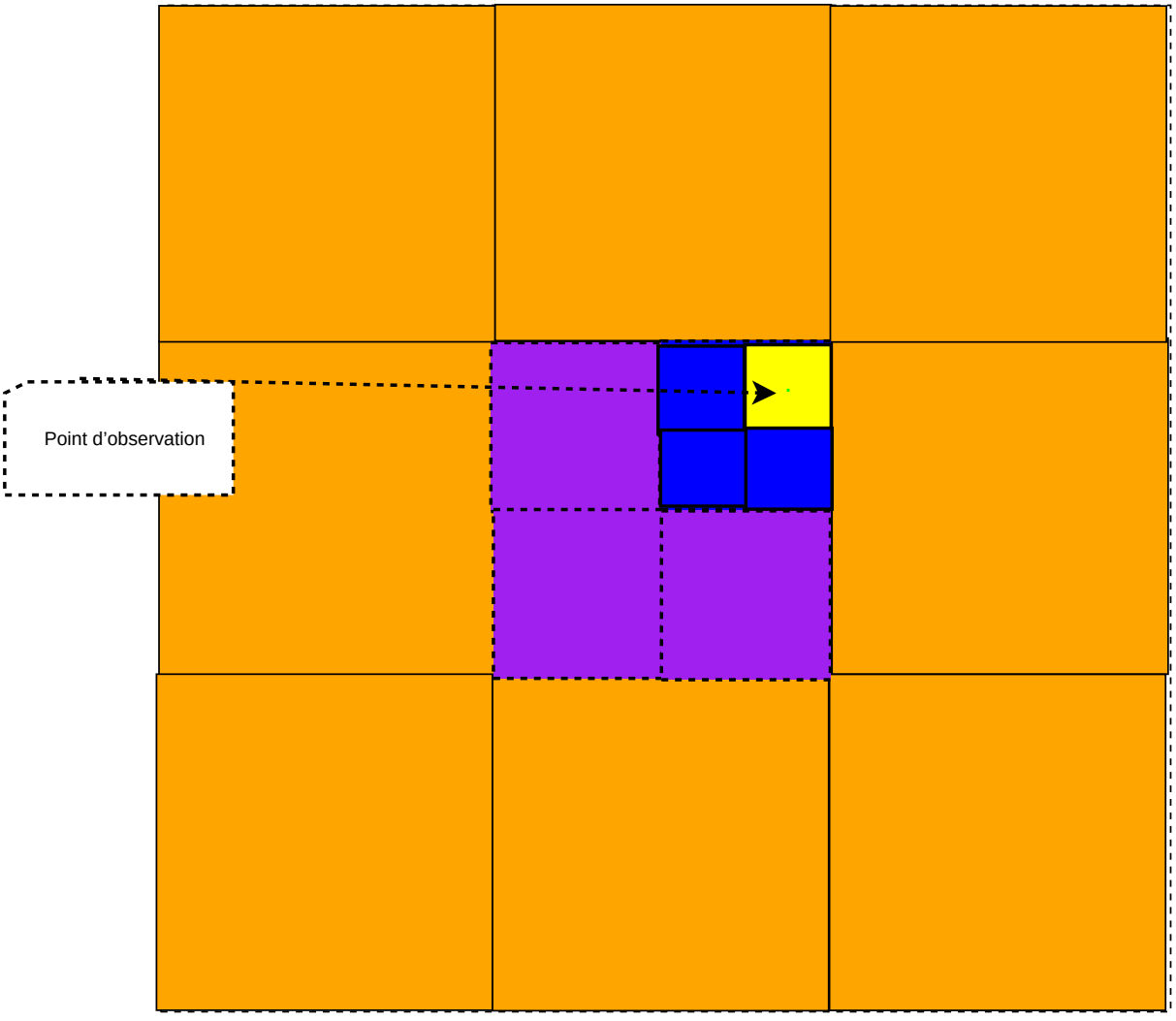

FIGURE 4.2 – Structure géo-spatiale hiérarchique

La description géo-spatiale d'une observation consiste à décrire l'endroit où celle-ci a été effectuée. La description est d'autant plus complète qu'elle tend vers un point unique identifié

par ses coordonnées de géo-référencement (latitude et longitude, ...). Ce point d'observation est contenu dans une zone caractérisée par des limites géographiques. Ces dernières peuvent être administratives ou naturelles. Une zone est décrite par ses limites administratives lorsqu'elle est identifiée par un nom de commune, département, région, pays ou continent. Lorsque la zone est décrite par les limites naturelles, alors, elle est identifiée par les coordonnées de géo-référencement maximales et minimales. Ces dernières sont les latitudes et longitudes maximales et minimales. Il faut noter qu'une zone administrative dispose de coordonnées de géo-référencement maximales et minimales.

Le géo-référencement par les coordonnées latitude et longitude du point d'observation suffit pour décrire le point d'observation géographique. Toutefois, une observation peut être décrite par une zone géographique plus ou moins étendue. Ce type de description est utilisé par exemple par certain fournisseurs pour protéger des espèces menacées en donnant la zone (pays ou le continent, ou des coordonnées limites de la zone) ou l'occurrence à été observée. De plus, lors de l'analyse des données, l'utilisateur peut arriver à faire des regroupements d'un ensemble d'observation d'une même zone. D'où l'intérêt d'avoir une organisation hiérarchique de la description géographique où une observation correspond à un point géo-référencé du niveau le plus bas de la hiérarchie.

Definition 4.2. Un nœud géo-spatial est défini par la concaténation de l'identifiant de son parent direct (zone conteneur supérieure hiérarchique) et son identifiant dans son groupe (éléments issues directement du même parent).

4.1.2 Spécificités des requêtes d'analyse de biodiversité

Plus de 80% des données d'observation partagées à travers le GBIF sont géo-référencées. Les cas d'usage de données de biodiversité étudiés à la section 3.1.3 nous permettent de constater que les analyses faites sur les données de biodiversité ont en commun certaines dimensions interrogées. Ces dimensions concernent la dimension taxinomique pour connaître les taxons à analyser et la dimension géo-patiale pour cadrer la zone de l'étude. Comme exemples, on peut citer la modélisation de la niche écologique de la chauve-souris, le calcul de la cooccurrence de deux espèces (ou genre) dans une région (ou un pays), la distribution des éléphants en Afrique, l'abondance des moustiques dans les régions tropicales, etc.

Cette particularité des requêtes d'analyse de biodiversité se traduit par des prédicats portant sur les dimensions taxinomiques (taxons ou nœuds taxonomiques) et géo-spatiales (zone ou nœud géo-spatial). Généralement, une requête est ciblée sur une petite portion de données concernant un ou plusieurs taxons et une zone. Les requêtes des utilisateurs peuvent être disjointes par les portions de données interrogées. Ce partitionnement implicite des requêtes d'analyse peut jouer un rôle important pour partitionner les données de biodiversité. C'est ainsi que nous utilisons

cette particularité des requêtes d'analyse de biodiversité que nous combinons avec la structure hiérarchique des données pour fragmenter les données de biodiversité.

4.1.3 Définition des fragments à la demande

Nous venons de voir que les données de biodiversité présentent une structure hiérarchique selon les dimensions taxinomiques et géo-spatiales et que les requêtes d'analyse en biodiversité concernent souvent ces deux dimensions (distribution, migration, co-occurrence, modélisation de niche écologique, etc.). De ce fait, nous supposons dans la suite qu'une requête dispose de prédicats concernant au moins une de ces deux dimensions. Les fragments traités par la requête sont donc définis par la conjonction des prédicats sur ces deux dimensions : pour chaque fragment nous avons un prédicat de sélection du taxon et un prédicat de sélection de la zone d'étude. Un prédicat taxinomique (respectivement géo-spatial) correspond à un nœud de la dimension taxinomique (respectivement géo-spatiale).

Definition 4.3. Un fragment est identifié par deux parties conjointes : nœud de la structure taxinomique et nœud de la structure géo-spatiale.

Exemples de fragments

Nous supposons que nous avons un attribut zone qui définit une zone géographique et un attribut espèce qui correspond à un nom d'espèce.

1. Les occurrences d'abeilles de la forêt amazonienne.
   Ce fragment est défini par le prédicat taxinomique *espece = 'abeille'* et le prédicat géographique *zone = 'amazonie'*. Il est identifié par le couple (abeille, amazonie)
2. Les occurrences de sardinelles sur les larges du Sénégal.
   Ce fragment est défini par le prédicat taxinomique *espece = 'sardinelle'* et le prédicat géographique *zone = 'senegal'*. Il est identifié par le couple (sardinelle, senegal)
3. Les occurrences de péguouin observées en Arctique.
   Ce fragment est défini par le prédicat taxinomique *espece = 'pinguin'* et le prédicat géographique *zone = 'arctique'*. Il est identifié par le couple (pinguin, Arctique).

Issus de la conjonction de deux dimensions hiérarchiques, les fragments de données de biodiversité présentent une structure hiérarchique. Cette caractérisation hiérarchique des fragments permet de déduire une relation de spécialisation/généralisation entre des fragments. Ce qui fait que bien que deux fragments n'aient pas les mêmes prédicats de définition, ils peuvent avoir une relation d'inclusion où le fragment le plus large inclut le plus petit. De ce fait, le fragment qui inclut l'autre peut être utilisé pour des traitements sur le fragment inclus. Ce qui favorise la réutilisation des fragments pour des requêtes qui ne les invoquent pas directement dans leurs

prédicats. Grâce aux catalogues taxonomiques et géo-spatiales, on peut retrouver les fragments qui généralisent (ou qui spécialisent) d'autres fragments.

Exemples de relations d'inclusion de fragments

Considérons les fragments suivants :
(sardinelle, senegal), (sardinelle, afrique), (insecte, espagne), (abeille, espagne), (abeille, europe).

1. La structure de la sémantique géo-spatiale permet de voir que le sénégal est inclus dans l'afrique. De ce fait, les sardinelles du sénégal font partie des sardinelles d'afrique. En d'autre termes le fragment correspondant aux sardinelles d'afrique inclut le fragment correspondant aux sardinelles du sénégal. Ceci fait que le fragment des sardinelles d'afrique peut être utilisé pour traiter une requête qui invoque le fragment des sardinelles du sénégal.
2. De la même façon le fragment correspondant aux sardinelles d'europe peut être utilisé pour traiter une requête qui invoque les sardinelles d'espagne.
3. La structure taxonomique permet de voir que les abeilles sont des insectes. D'où les abeilles d'espagne sont dans le fragment des insectes d'espagne.

4.1.4 Fragmentation à la deemande

La création d'un fragment se fait à la demande. Pour cela, à la réception d'une requête, nous extrayons les prédicats taxonomiques et géo-spatiales. Le catalogue de la sémantique taxonomique nous permet de connaître le nœud taxonomique invoqué. De la même façon, le catalogue de la sémantique géo-spatiale nous permet de connaître le nœud géo-spatial invoqué. Ainsi, on peut créer créer, identifier, localiser le fragment ou encore déterminer les fragments pour lesquels il a des relations de généralisation/spécialisation.

4.2 Placement dynamique des fragments à la demande

La stratégie de placement a pour objectif de garantir la disponibilité des données traitées en les rapprochant des utilisateurs. Notre stratégie de placement des données distingue les deux phases du processus de répartition à savoir la création (ou placement) initiale de fragments et la réplication de données.

4.2.1 Création initiale des fragments

Le création initiale de fragments est la première étape qui suit le processus de fragmentation. Elle consiste à insérer dans les sites locaux des nouvelles données issues du GBIF qui sont sollicitées par une requêtes. Cette étape correspond à une réplication partielle à la demande

ou encore de caching de données du GBIF vers les sites locaux. Les fragments sont déterminés grâce aux prédicats taxinomiques et/ou géographiques de la requête. De ce fait, les bases locales sont alimentées au fur et à mesure que les requêtes invoquent des fragments manquants (qui n'existent pas sur un site local). D'où le concept de création à la demande. Le site devant héberger un fragment est déterminé par l'optimisation basée sur le coût de la requête décrite à la section 6.3. Lorsqu'un fragment existe déjà dans un site local, alors ce site est utilisé en remplacement du portail GBIF pour la lecture ou le traitement du fragment. Une telle réplication à la demande, permet de minimiser les coûts de communication avec le GBIF tout en favorisant le parallélisme des accès et des traitements à travers les sites locaux. Un processus de rafraîchissement des fragments décrit dans la section de gestion des mises à jour 4.5 est utilisé pour maintenir la cohérence des fragments locaux par rapport à la base complète du GBIF. Il importe de noter que cette approche est différente des techniques de caching classiques ou des vues matérialisées qui stockent le résultat de la requête initiale. En effet, notre approche se sert dees prédicats taxonomiques et géographique uniquement pour décider des données (correspondantes aux fragments impliqués) à stocker dans la base. De ce fait, la requête initiale peut être traiter avec ces nouvelles données.

4.2.1.1 Réplication dynamique basée sur le coût

Dans l'objectif de garantir un temps de réponse borné pour les requêtes, la réplication d'un fragment sollicité qui se trouve sur un site surchargé peut jouer un rôle important. Puisque la charge d'un site est dynamique, nous proposons d'intégrer la décision de répliquer un fragment dans le mécanisme d'optimisation de requêtes (voir la section 6.3). L'objectif est de placer une réplique du fragment sur un site qui convient pour garantir un temps de réponse borné pour les requêtes qui interrogent le fragment. Ces nouvelles répliques pourront être utilisées lors des traitements des requêtes ultérieures. La réplication de fragment est coordonné par le traitement de requêtes (voir la section 5.2).

4.3 Capacité de stockage limitée : une stratégie de remplacement adaptée

La réplication dynamique des données engendre un coût de stockage croissant. Or, la capacité de chaque site est limitée. Ainsi, nous adoptons une stratégie de remplacement qui adapte le contenu de la base en fonction de sa capacité pour garantir la disponibilité des données sollicitées. Pour cela, la stratégie conserve seulement dans la base de données les fragments les plus populaires en supprimant ou déplaçant les fragments les moins populaires qui ne sont pas utilisés par un traitement (en cours ou en attente), lorsque le contenu dépasse la taille autorisée.

La popularité d'un fragment correspond au nombre de fois qu'il a été demandé dans un intervalle de temps donné. L'intervalle de temps correspond au délai passé depuis le dernier remplacement qui a été effectué sur le site jusqu'à l'instant courant. Chaque site compte le nombre d'accès local au fragment et renvoie l'information au GAM sur demande. Lors du remplacement, le site concerné demande au GAM les popularités de ses fragments locaux. Pour chaque fragment, le GAM demande les nombres d'accès aux sites qui en disposent. Le cumul des résultats obtenus constitue la popularité du fragment.

Pendant le remplacement, un fragment impopulaire est supprimé si d'autres bases voisines en disposent. Dans le cas contraire, il est déplacé vers un site ayant de la place de stockage disponible.

Le remplacement est déclenché lorsque la taille de la base atteint un seuil égal à un certain pourcentage (Smax) de la capacité du site. Il enlève les fragments les moins populaires jusqu'à ce que la taille courante devienne inférieure à un seuil (Smin). Les valeurs des seuils Smax et Smin sont choisies de sorte que les données populaires soient conservées dans la base et que le remplacement ne soit pas trop fréquent. En effet, plus la valeur de Smin est petite plus la base locale est vidée pendant un remplacement. Ceci présente l'avantage de réduire la fréquence des remplacements. Cependant un problème de disponibilité de données populaires qui pourraient être supprimées de la base, se pose. En outre, une valeur petite de Smax réduit l'utilisation des ressources de stockage du site. Par ailleurs, une valeur maximale de Smax (100 % de la capacité totale) ne permet pas d'insérer de nouvelles données lorsque le seuil n'est pas atteint et que la taille des données à insérer est supérieure à l'espace de stockage restant. Le choix de ces deux seuils doit garantir la disponibilité des données populaires et minimiser les coût des remplacements (suppression ou transfert).

Il importe de noter aussi que cette stratégie de remplacement qui combine LFU (popularité de chaque fragment) et LRU (depuis le dernier remplacement), permet d'adapter le degré de réplication de chaque fragment en fonction de son taux d'utilisation récent. En effet, les fragments les plus sollicités ne seront pas supprimés et seront répliqués lorsque les sites qui les hébergent sont surchargés. Alors que les degrés de réplication des fragments les moins sollicités sont davantage réduits lorsque les sites les abritent sont remplis avec des fragments sollicités.

4.4 Indexation et localisation des fragments

La répartition des données à travers les sites aboutit à un schéma de placement. Le schéma de placement désigne la façon dont les fragments sont alloués aux sites. Nous proposons un mécanisme d'indexation et de localisation des données à travers l'architecture décentralisée. L'objectif de ce mécanisme est de localiser rapidement les données impliquées lors de l'évaluation

des requêtes. Ceci est un point essentiel pour optimiser les performances et gérer le passage à l'échelle. Ainsi, nous proposons une indexation hybride qui est composée d'un catalogue local au niveau de chaque site et un catalogue global qui est hébergé au niveau du GAM. Ces index sont consultés pendant l'analyse de la requête pour localiser les données nécessaires au traitement.

4.4.1 Indexation hybride des données

Chaque site dispose d'un catalogue local qui indexe tous les fragments hébergés dans la base de données locale et les fragments sollicités par les requêtes qui lui sont posées. Cet index local n'a pas une connaissance globale sur le contenu de tous les sites. Il est complété en cas de nécessité par le catalogue global du GAM qui dispose des informations sur les contenus de tous les sites. Ainsi, si l'index global permet de garder l'information de localisation de tous les fragments à travers le système, les index locaux visent à décharger le GAM des tâches de localisation de données.

La figure 4.3 illustre notre mécanisme de combinaison des index locaux (en vert) au niveau des sites (rectangle bleu) et d'un index global (en vert) au niveau du GAM (rectangle violet). Dans cet exemple, chaque site garde les emplacements de tous les fragments qu'il contient. Le GAM garde les emplacements de tous les fragments disponibles à travers le système.

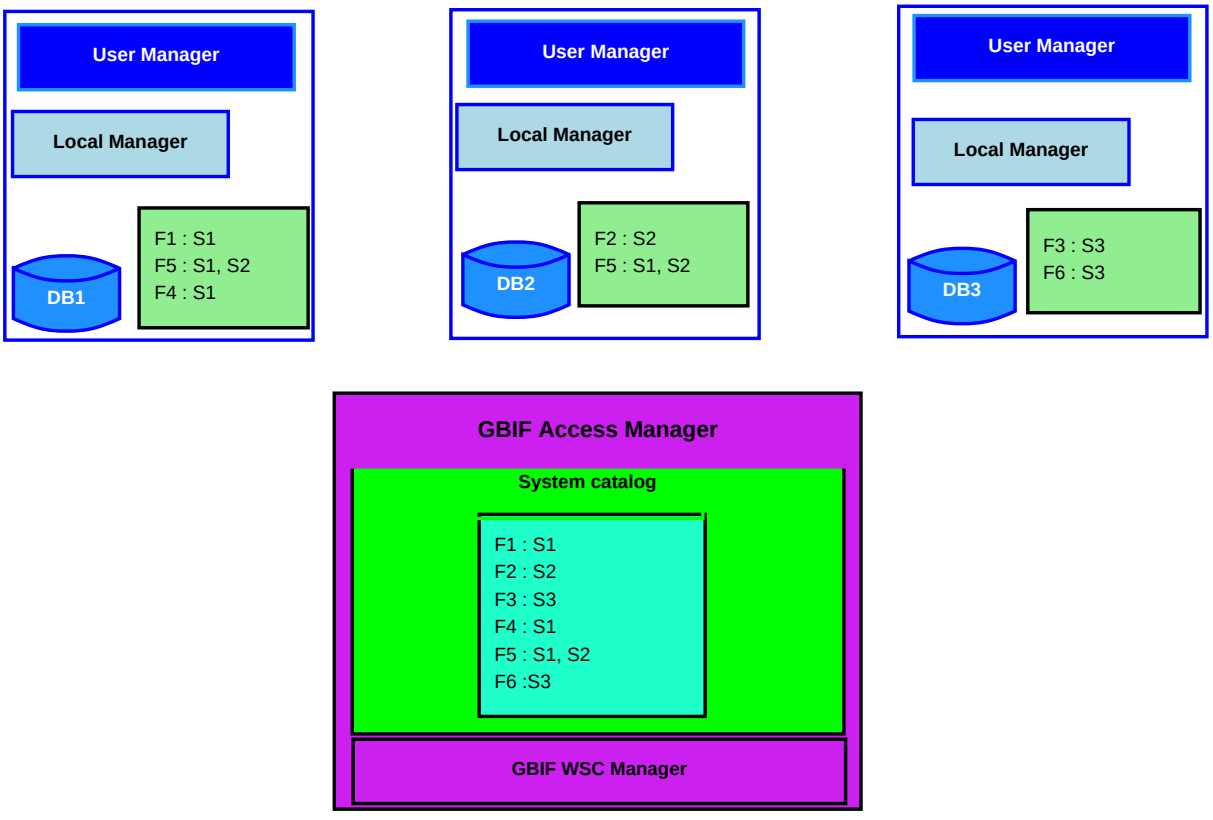

FIGURE 4.3 – combinaison d'index locaux et d'index global

Dans l'optique de retrouver le plus rapidement possible les localisations des fragments à traiter, sans pour autant surcharger le GAM et le réseau, chaque site, en plus des fragments de la base locale, indexe également les fragments sollicités par les requêtes qui lui sont envoyées. Pour cela, lorsque le gestionnaire de requêtes (User Manager) ne connait pas les emplacements d'un fragment au cours de l'analyse d'une requête, il interroge le catalogue global du GAM, qui lui indique les emplacements des différentes répliques du fragment recherché. Cette information est insérée dans l'index local et est utilisée dans l'analyse des prochaines requêtes qui invoquent le fragment.

4.4.2 Localisation des données

L'analyse de la requête permet de connaître les fragments impliqués. Le gestionnaire de requêtes (User Manager) consulte l'index local pour localiser chaque fragment de la requête. Lorsqu'il existe des fragments qu'on ne peut pas à localiser via l'index local, alors le gestionnaire de requêtes envoie les demandes de localisations de ces fragments au GAM qui consulte son index global. Ce qui lui permettra de renvoyer les emplacements des fragments concernés au gestionnaire de requêtes demandeur.

La figure 4.4 montre un exemple de recherche des fragments F1 et F3 impliqués par une requête. Pour localiser les fragments F1 et F3 de la requête, le gestionnaire de requêtes consulte l'index local (2) qui lui fournit les emplacements de F1 seulement (3). N'ayant pas de réponse pour F3, le gestionnaire de requêtes envoie une demande de localisation de F3 au GAM (4) qui lui indique les emplacements de celui-ci (5). Par la suite, le gestionnaire de requêtes devra ajouter dans l'index local une entrée pour F3 afin de répertorier les emplacements de celui-ci.

Ce mécanisme d'indexation et de localisation qui combine un catalogue global et des indexes locaux permet à la fois de garantir la cohérence des informations de localisation des fragments et d'éviter les congestions pour les recherches.

4.4.3 Gestion de la cohérence des index

Le GAM est informé de toutes les modifications et mouvements apportés aux fragments des sites locaux afin qu'il mette à jour le catalogue global. En effet, après avoir transmis un nouveau fragment venant du GBIF au site demandeur, il complète le catalogue global en ajoutant une entrée pour le nouveau fragment. En outre, le GAM est informé de toutes les réplications de fragments entre les sites locaux produites pendant l'exécution des requêtes. De cette façon, le catalogue global enregistre les localisations courantes des répliques de chaque fragment.

Afin de synchroniser les informations de localisation d'un fragment dans l'index local par rapport à d'éventuels changements le concernant sur l'index global, toute modification apportée sur les

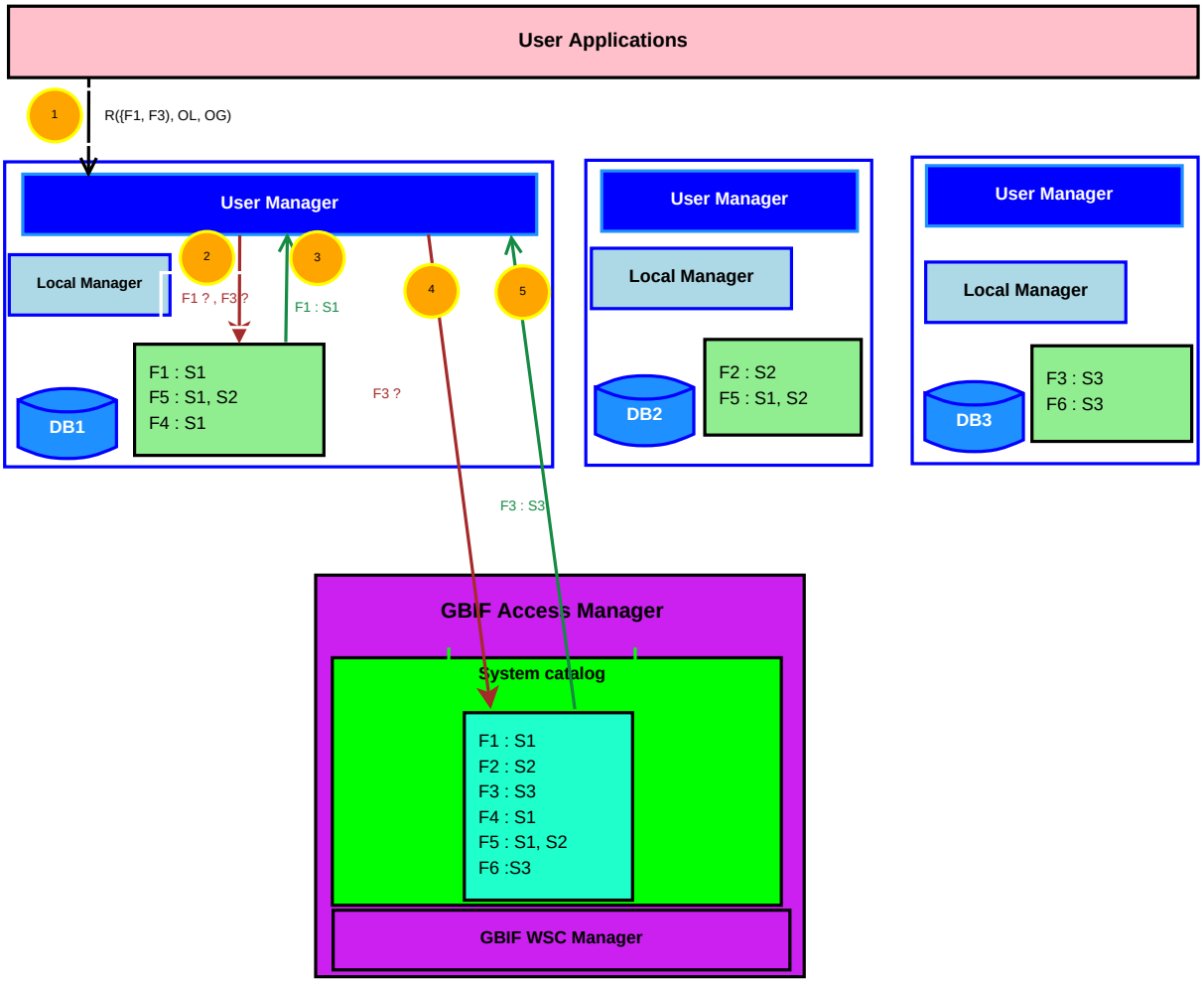

FIGURE 4.4 – Localisation des données

emplacements du fragment est propagée immédiatement vers les sites qui l'indexent pour qu'ils rafraîchissent leurs catalogues locaux.

Pour éviter qu'un même fragment soit répliqué simultanément par deux initiateurs, la réplication est synchronisée via l'index global. Pour cela, lorsqu'un site initie la réplication d'un fragment, il informe le GAM qui verrouille alors toute autre possibilité de réplication du fragment. Le verrou est enlevé lorsque la réplication est effective. Avant de démarrer une réplication, le site destinataire doit disposer d'une autorisation auprès du GAM. Une réplication est autorisée lorsque le fragment concerné n'est pas verrouillé. Lorsqu'un site souhaite répliquer un fragment qui est verrouillé, alors il recalcule le plan d'exécution des opérations locales concernant le fragment en question en considérant qu'une réplique du fragment serait sur le site destinataire de la réplication.

4.4.4 Coordination des réplications

Pour réduire les coûts de la réplication pour une donnée, la réplication doit être validée par le GAM. Pour cela, lorsqu'un site doit traiter une donnée qu'il doit répliquer auparavant, il demande une autorisation de réplication au GAM. Ce dernier autorise la réplication lorsqu'il n'existe de réplication de la donnée concernée. Dans le cas, contraire il refuse la réplication au site demandeur en lui précisant les destinations des réplications en cours de la donnée. Le site demandeur calcule un plan d'exécution du traitement (sous-requête) sur la donnée en supposant que la donnée existe sur leurs destinataires en tenant compte de la date de disponibilité de la donnée. Si le résultat de ce calcul nécessite que le fragment soit répliqué en local, alors le site soumet à nouveau une demande d'autorisation de réplication du fragment auprès du GAM. Lorsque le GAM reçoit une deuxième demande d'autorisation de réplication du même fragment venant du même site, alors, l'autorisation est validée s'il n'existe aucune réplication en cours du fragment. Sinon, le GAM vérifie si d'éventuelles réplications du fragment ont été déclenchées entre temps depuis la dernière demande de réplication du fragment du site qui a été refusée. Lorsque des réplications du fragment n'ont pas été autorisées, alors la nouvelle demande est satisfaite. De ce fait, le site peut démarrer sa réplication. Par contre, lorsque des réplications du fragment ont été déclenchées, alors, le GAM refuse la nouvelle demande au site et lui précise les destinations des nouvelles réplications. Dans ce cas, le site reprend la même procédure que précédemment. Ce processus est répété jusqu'à ce le fragment puis être traité sur les nouvelles destinations ou que la réplication soit autorisée.

Pour éviter qu'un fragment soit répliqué simultanément par deux initiateurs, la réplication est synchronisée via l'index global. Pour cela, lorsqu'un site initie la réplication d'un fragment, il informe le GAM qui verrouille alors toute autre possibilité de réplication du fragment. Le verrou est enlevé lorsque la réplication est effective. Avant de démarrer une réplication, le site destinataire doit disposer d'une autorisation auprès du GAM. Une réplication est autorisée lorsque le fragment concerné n'est pas verrouillé. Lorsqu'un site souhaite répliquer un fragment qui est verrouillé, alors il recalcule le plan d'exécution des opérations locales concernant le fragment en question en considérant qu'une réplique du fragment serait sur le site destinataire de la réplication.

4.5 Mises à jour asynchrones des fragments locaux

Les données de la base complète du portail du GBIF peuvent être mises à jour. La solution décentralisée doit garantir la cohérence des fragments locaux (répliques disponibles dans le système décentralisé) par rapport à la base complète du GBIF. Nous proposons un mécanisme de propagation des mises à jour apportées aux fragments du GBIF ayant des copies dans les

bases locales. Cela consiste à interroger le GBIF momentanément pour obtenir les données ajoutées ou modifiées dans chaque fragment existant dans les sites locaux. En cas de mises à jour des fragments concernés aux GBIF, il remplace toutes les répliques locales de ces fragments modifiées par leurs nouvelles versions. Afin de ne pas ralentir le traitement des requêtes, les interrogations de mises à jour se font en dehors des périodes d'exploitation. Ainsi, les requêtes accèdent à un cliché des données (snapshot) suffisaament récent pour convenir aux utilisateurs. Dans le cas de l'analyse de données de biodiversité, un cliché demeure récent pendant plusieurs jours. Ce mécanisme de propagation des mises à jour n'est pas coûteux dans la mesure où les mises à jour effectuées dans la base du GBIF sont généralement des ajouts (insert) et que les modifications (update) et suppressions (delete) sont très rares. De ce fait, les mises à jour des fragments locaux seraient rares puisque les fragments primaires du GBIF sont rarement modifiés.

4.6 Conclusion

Dans ce chapitre, nous proposons une stratégie de répartition dynamique des données de biodiversité à travers le système décentralisée. Nous présentons notre approche de partitionnement dynamique des données de biodiversité en fonction des prédicats des requêtes. Nous exploitons la structure hiérarchique des données de biodiversité et les spécificités des requêtes d'analyse pour fragmenter les données. Notre approche de placement et réplication dynamique à la demande est contrôlée par l'optimisation de requêtes et est différente des solution de caching coopératives classiques. Nous complétons la répartition dynamique des données avec une stratégie de remplacement de donnée à travers une base locale lorsqu'elle est pleine, pour y préserver de l'espace de stockage pour les nouvelles données et garantir la disponibilité des données sollicitées par les requêtes. Nous proposons aussi, un mécanisme d'indexation hybride et de localisation des données qui permet de retrouver rapidement les emplacements d'un fragment sans surcharger le réseau et le GAM. Enfin, nous proposons une approche de gestion des mises à jour des données qui se trouvent dans le système décentralisé pour garantir leur cohérence par rapport aux données complètes du portail du GBIF.

Chapitre 5

Traitement de requêtes

Dans ce chapitre, nous proposons un mécanisme d'exécution des requêtes à travers l'architecture décentralisée que nous avons présentée à la section 3.3, pour favoriser la disponibilité du système vis-à-vis des utilisateurs et le passage à l'échelle des fonctionnalités. Le contexte de notre solution présente la particularité que toutes les données sont initialement sur un seul site, le portail GBIF. De ce fait, notre stratégie de traitement de requêtes contrôle aussi la répartition des données à travers les sites locaux. Intuitivement, nous devons atteindre un état tel que tous les sites participent au traitement des requêtes pour éviter que certains sites soient surchargés au moment où d'autres sont libres. Cela peut nécessiter de répliquer certaines données sollicitées fréquemment.

Nous présentons dans les sections suivantes le modèle de requêtes d'analyse de données de biodiversité, puis le mécanisme d'exécution de la requête, ensuite, le traitement concurrent de plusieurs requêtes et, enfin, le partage d'information entre les sites.

5.1 Modèle de requête

Une requête, notée $R(O, F)$, est caractérisée par un ensemble d'opérations O qui doivent être appliquées sur un ensemble de fragments F. Nous notons O_k un élément de O et F_j un élément (un fragment) de F. Nous distinguons deux types d'opérations :

1. *Opération locale* . Les opérations communes à tous les F_j et qui peuvent être exécutées en parallèle sur chaque F_j indépendamment des autres fragments. Nous les notons OL (pour Opérations Locales aux fragments).
   $\mathrm{OL} = \{\mathrm{O}_k \in \mathrm{O} \ / \ \mathrm{O}_k$ applicable à tous les $\mathrm{F}_j\}$.
2. *Opération globale* . Les opérations qui ne peuvent pas être exécutées en parallèle sur chaque F_j mais de façon globale sur les résultats des traitements OL. Nous les notons OG (pour

Opérations Globales aux fragments).

OG = {$O_k \in O$ / O_k non-applicable à tous les F_j}.

Definition 5.1. *Traitement OL*

Un traitement local (dénoté traitement OL) est l'ensemble des opérations de la requête qui sont locales à chaque fragment.

Definition 5.2. *Traitement OG*

Un traitement global (dénoté traitement OG) est l'ensemble des opérations de la requête qui sont exécutées sur les résultats des traitements OL de la requête.

Nous illustrons ce modèle de requête à travers la figure 5.1 où D_1 et D_2 correspondent respectivement aux densités des plantes (family = 'Plantae') et des abeilles (family = 'Apidae') dans la zone Z_x délimitée par min_decimallatitude = -10, max_decimallatitude = 20, min_decimallongitude = 30 et max_decimallongitude = 60. Dans cet exemple *Pave* correspond à une table qui définit le maillage de la zone d'étude Z_x. Le maillage définit le découpage de la zone en un ensemble de compartiments (mailles) disjoints. Il permet d'avoir une vision plus ou moins détallée de l'étude.

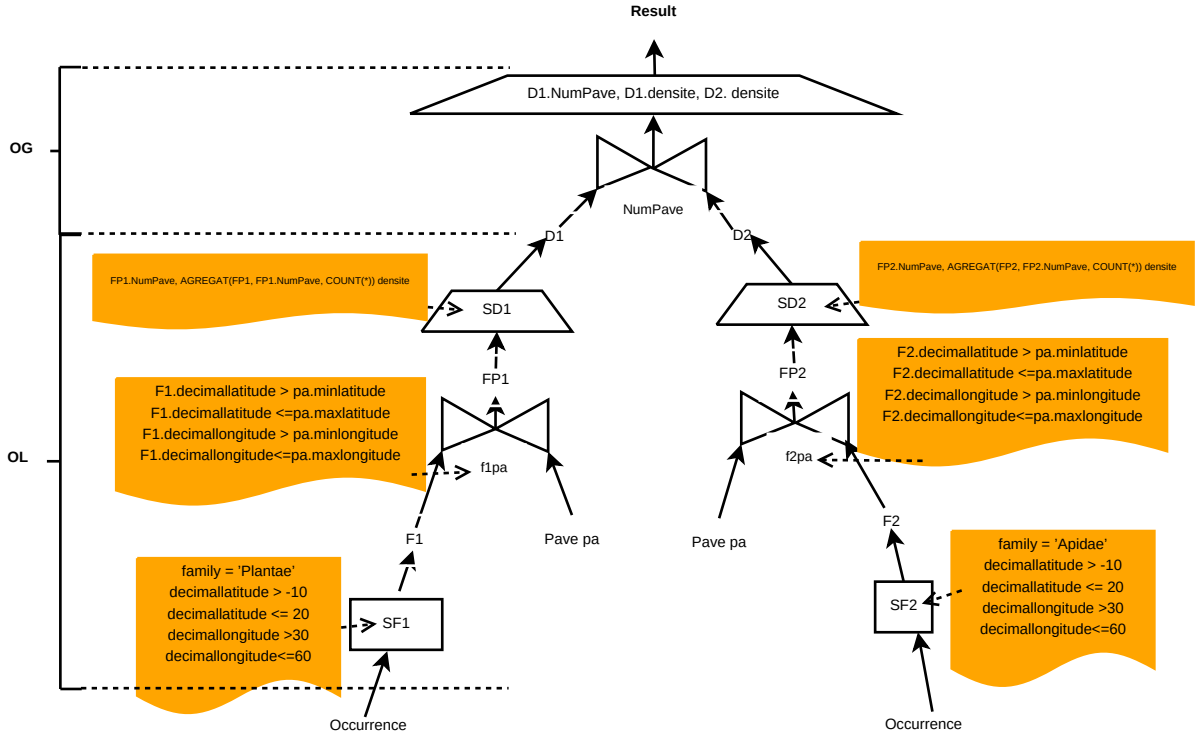

FIGURE 5.1 – Modèle de requête d'analyse de biodiversité

Nous notons une requête par $R(\{OL, OG\}, F)$ et considérons que le coût d'un traitement (OL ou OG) est proportionnel à la quantité de données impliquées. Nous supposons que pour la

plupart des requêtes de biodiversité, le coût des traitements OL est plus important que celui des traitements OG. En effet, les traitements OL sont des sélections suivies d'agrégations et de jointures et impliquent des traitements complexes, alors que les traitements OG sont des unions ou des jointures sur les résultats des OL (petites quantités de données).

Par exemple, le calcul des cooccurrences de deux ou plusieurs espèces se décompose comme suit :

i) Traitement OL : créer le maillage de la zone d'étude et déterminer dans quelle maille est localisée chaque occurrence de l'espèce étudiée. Les occurrences de la même espèce sont dénombrées pour chaque maille afin d'obtenir les valeurs de densité.

ii) Traitement OG : pour chaque maille, les valeurs de densité des différentes espèces sont combinées pour obtenir les valeurs de cooccurrence.

Dans cet exemple, les traitements OL s'avèrent plus coûteux que les traitements OG. C'est un cas d'usage représentatif des requêtes généralement posées pour analyser les données de biodiversité. Les études de migration, de distribution, la modélisation de niche écologique, les relations de proie/prédation, nécessitent des calculs de cooccurrence et de densité d'espèces. Nous visons à apporter des solutions efficaces pour ces cas d'usage, en exploitant la possibilité d'évaluer en parallèle les traitements OL.

5.2 Mécanisme d'exécution de requête

La figure 5.2 montre les principales étapes de l'exécution d'une requête. A la réception d'une requête d'utilisateur, le gestionnaire de requêtes (UM) analyse la requête pour déterminer les fragments nécessaires et leurs relations d'inclusion avec d'autres fragments qui seraient suffisants pour remplacer un ou plusieurs fragments de la requête. Grâce à l'index local et éventuellement à l'index global, il détermine les localisations des fragments impliqués. Ces informations de localisation des fragments sont complétées avec les informations sur les charges des sites pour optimiser la requête. L'optimisation de la requête produit un plan d'exécution qui consiste en l'ensemble des opérations et fragments à traiter sur chaque site. Ce plan est calculé grâce à un algorithme d'optimisation que nous verrons plus tard. Par la suite, nous distinguons respectivement le UM qui a reçu, analysé et optimisé la requête et le UM désigné pour coordonner le traitement, par QUM (Queried UM) et CUM (Coordinator UM). Le plan d'exécution peut engendrer des réplications de données entre des sites locaux ou un accès à la base du GBIF pour télécharger les données manquantes. Le CUM envoie à chaque UM d'un site impliqué sa sous-requête avec ses éventuelles demandes de réplication. Celui-ci coordonne les éventuelles réplications et demandes GBIF (via le GAM) puis traite sa sous-requête avec son gestionnaire local (LM) et envoie les résultats partiels au CUM. A la réception de tous les résultats partiels,

le CUM démarre le traitement des opérations globales de la requête à l'aide du LM du même site et du SGBD local.

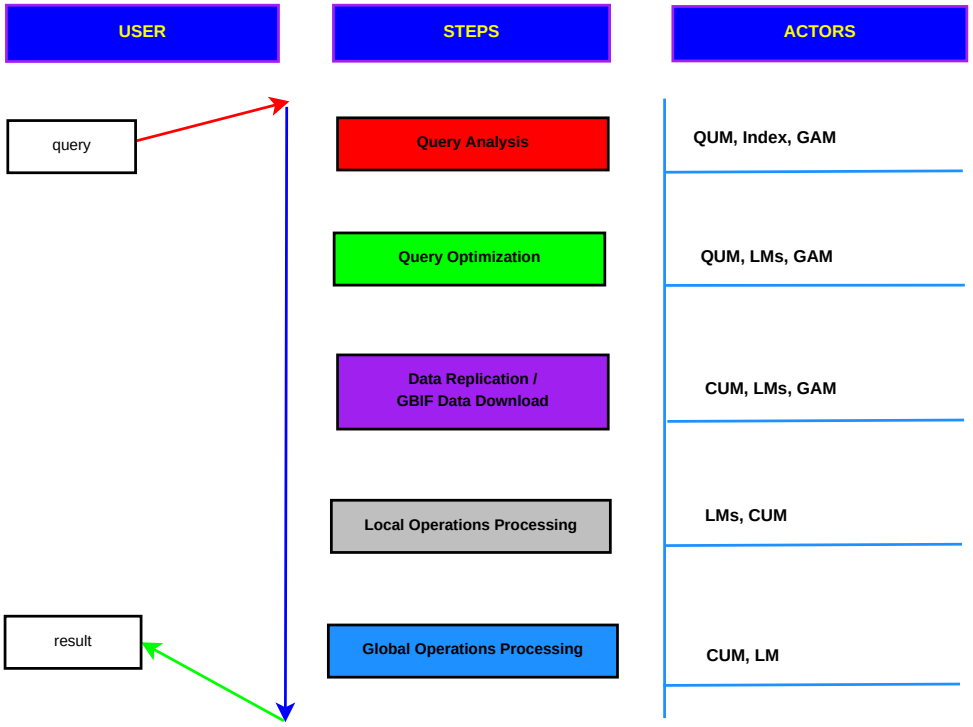

FIGURE 5.2 – Etapes et acteurs du traitement de la requête

Nous allons détailler les différentes étapes de l'exécution d'une requête.

5.2.1 Analyse de la requête

L'analyse de la requête est illustrée à la figure 5.3 avec ses étapes successives. Elle consiste à

1. déterminer l'ensemble des fragments F de données invoquées par la requête.
2. Cet ensemble est constitué de fragments (F_i) explicités directement dans les prédicats (p_i) de la requête. Ces fragments (F_i) peuvent être constitués d'un ensemble de (sous) fragments (F_{ij}) non explicités dans la requête.
3. Pour chaque fragment de la requête, on peut connaitre les éventuels fragments qui le composent grâce aux relations d'inclusion (de contenance) invoquées à la section 4.1. Lorsqu'un fragment est composé de sous-fragments (F_{ij}), ces derniers sont utilisés dans la suite de l'analyse qui consiste à trouver leurs localisations.
4. Le QUM consulte son index local pour connaître au moins les fragments disponibles dans la base de données locale et éventuellement des fragments distants.
5. Si l'index local ne permet pas de localiser tous les fragments issus de l'analyse à travers les sites locaux,

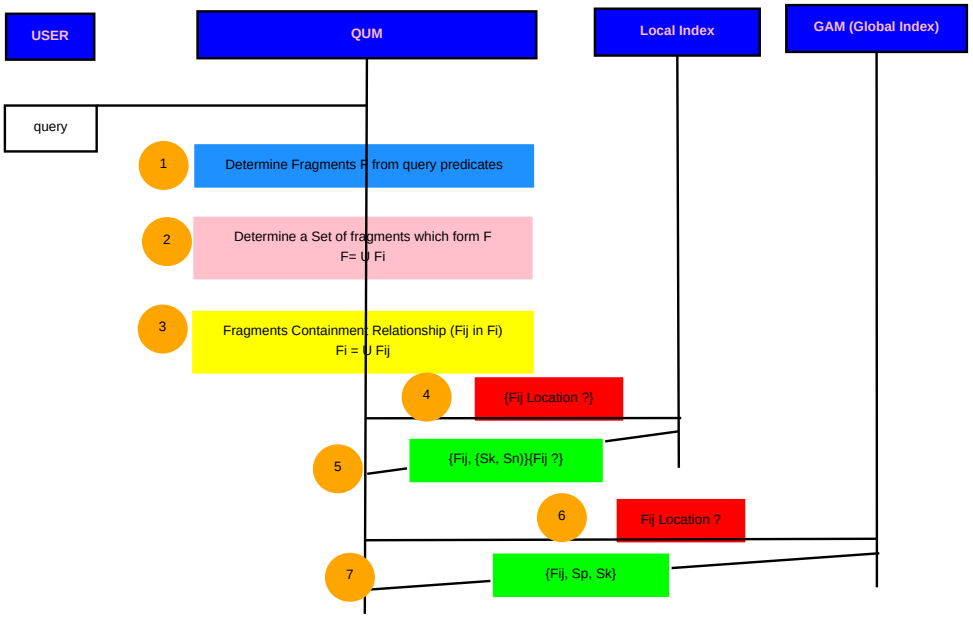

FIGURE 5.3 – Analyse de la requête

6. alors le QUM consulte l'index global du GAM. Si malgré la consultation de l'index global, on ne retrouve pas la localisation d'un fragment, alors ce dernier est considéré comme manquant et donc sera téléchargé depuis le portail du GBIF.

Pour ce qui concerne les opérations, on suppose que les opérations OL et OG sont explicitées dans la requête. De ce fait, l'analyse se concentre sur les données et produit une liste de fragments élémentaires impliqués par la requête et leurs localisations.

5.2.2 Optimisation générale de la requête

L'optimisation de la requête est basée sur la localisation des fragments retournés par l'analyse. Ces informations de localisation des fragments sont combinées avec les informations sur les charges des sites pour appliquer l'algorithme d'optimisation qui aboutit au plan d'exécution de la requête en tenant compte de la contrainte de temps de réponse borné. Les informations de charge de chaque site sont envoyées par le LM concerné sur demande du QUM. Puisque le LM dispose des statistiques sur la durée locale des traitements en attente, il peut connaître la charge du site local en faisant la somme des durées des traitements en attente sur son site. L'optimisation de la requête détaillée à la section 6.3 du chapitre 6, produit deux types de plans : un plan d'exécution, composé d'une liste de couples (Fragment, Site) pour indiquer le site désigné pour traiter chaque fragment de la requête et un plan de réplication constitué de triplets (Fragment, site destinataire, site source) pour indiquer qu'en cas de réplication de fragment, la source et la destination de la nouvelle réplique. La figure 5.4 illustre l'optimisation générale de la requête.

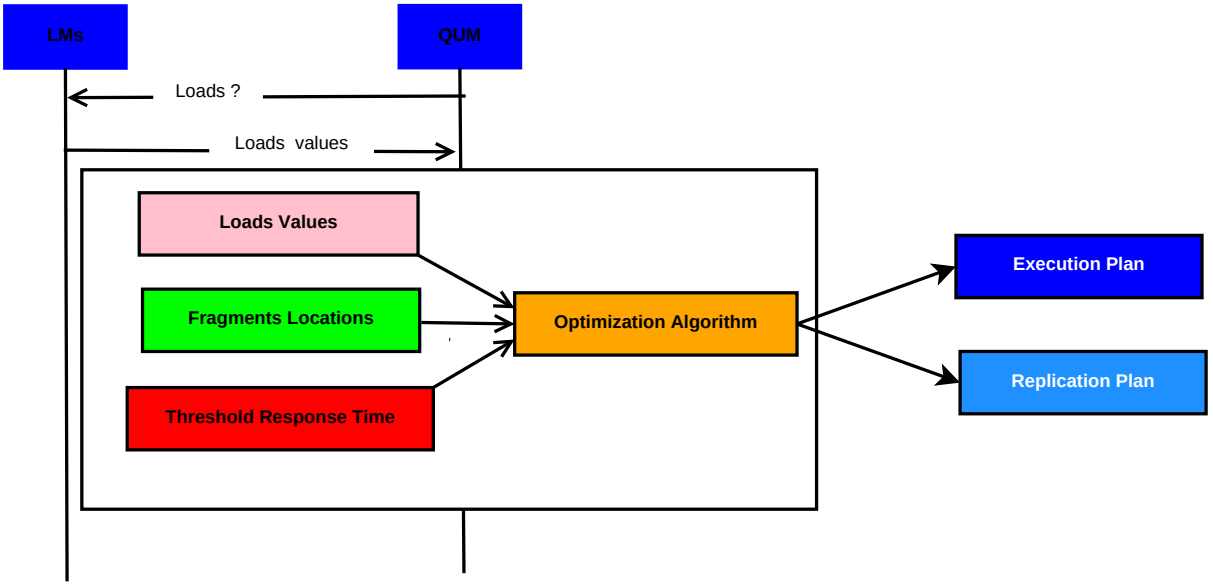

FIGURE 5.4 – Optimisation générale de requêtes

5.2.3 Réplication de données

Le processus de réplication de données est illustré à travers la figure 5.5. Après avoir analysé et optimisé la requête, le QUM envoie le plan d'exécution au CUM (le QUM et le CUM peuvent être sur le même site). A la réception des plans d'exécution et de réplication, le CUM envoie à chaque site impliqué sa sous requête. Lorsque qu'un site reçoit une sous-requête portant sur un fragment dont il ne dispose pas, alors il coordonne la réplication en envoyant la requête de sélection du fragment à la source qui la traite et lui renvoie les résultats (répliques). A la réception des répliques, il les insère dans la base de données locale et démarre le traitement local de la sous-requête.

Avant qu'un site ne démarre une réplication, il demande une autorisation auprès du GAM qui vérifie l'état du fragment concerné. Le fragment peut être verrouillé ou autorisé en réplication. Un fragment est verrouillé en réplication lorsqu'il est déjà en cours de réplication. De cette façon à un instant donné seule une réplication est possible pour un fragment donné. Lorsqu'un fragment est verrouillé en réplication, le GAM informe le demandeur de la destination de la réplication en cours afin qu'il calcule un nouveau plan d'exécution pour ce fragment (uniquement) en tenant compte de la nouvelle réplique.

5.2.4 Réplication à la demande de données du GBIF

La réplication à la demande de données à partir du GBIF est illustrée à travers la figure 5.6. Lorsque les données n'existent dans aucun site local, il est nécessaire de répliquer les données à

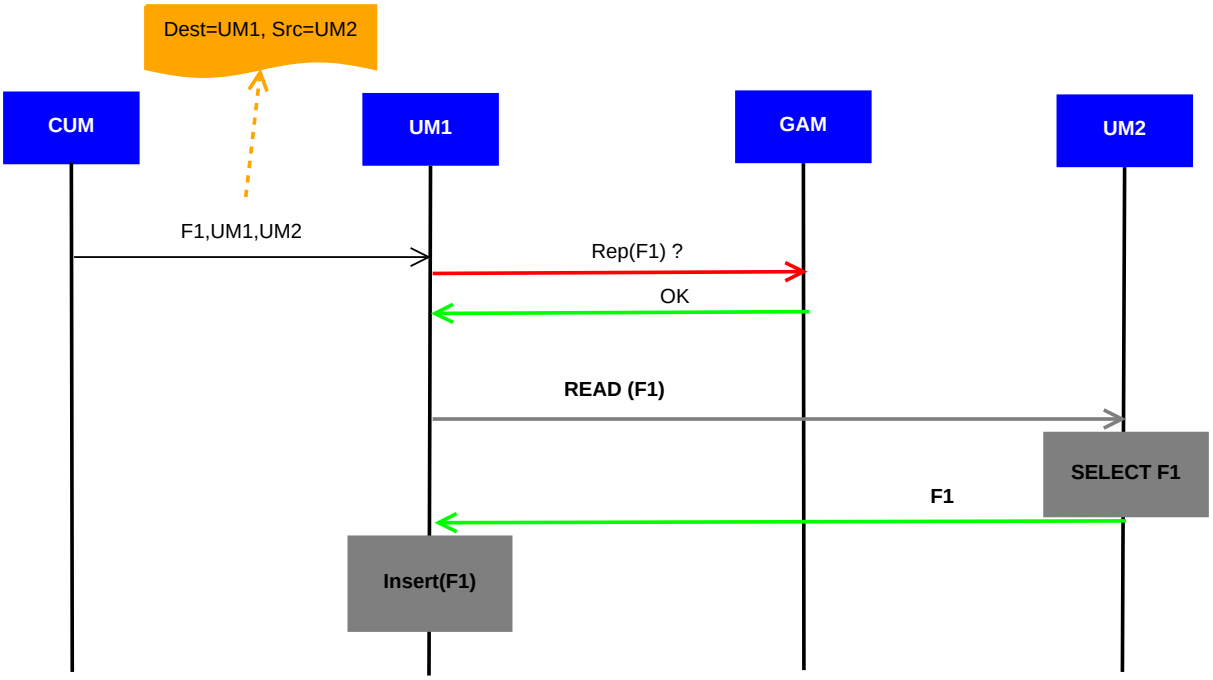

FIGURE 5.5 – Processus de réplication de données

partir du GBIF et de les insérer dans les bases de données des sites locaux avant leur traitement. Un mécanisme similaire au processus de réplication est adopté. Le destinataire du nouveau fragment délègue au GAM le téléchargement des données à partir du GBIF. Le GAM formule les appels de service web correspondants pour chaque fragment et les soumet au portail GBIF. A la réception des données venant du GBIF, le GAM les redirige vers le destinataire.

Lorsqu'un site reçoit un fragment venant du GBIF via le GAM, il l'insère dans la base de données locale et démarre le traitement de ses sous-requêtes. Afin d'éviter que des appels redondants d'un même fragment se produisent, le GAM marque déjà dans son index que le fragment est en cours de téléchargement du GBIF. Lorsque plusieurs sites destinataires demandent le même fragment, alors le GAM envoie une copie à chaque destinataire.

5.2.5 Traitement de la requêtes

A la réception du plan d'exécution, le CUM formule les sous-requêtes avec les éventuelles demandes de réplication en fonction des sites concernés pour le traitement de chaque fragment. Pour formuler les sous-requêtes, le plan d'exécution de la requête, donne la liste des fragments à traiter au niveau de chaque site impliqué et les opérations OL correspondantes. Ainsi à chaque site impliqué, correspond une sous-requête contenant l'ensemble des fragments qui y seront traités et les opérations OL de la requête.

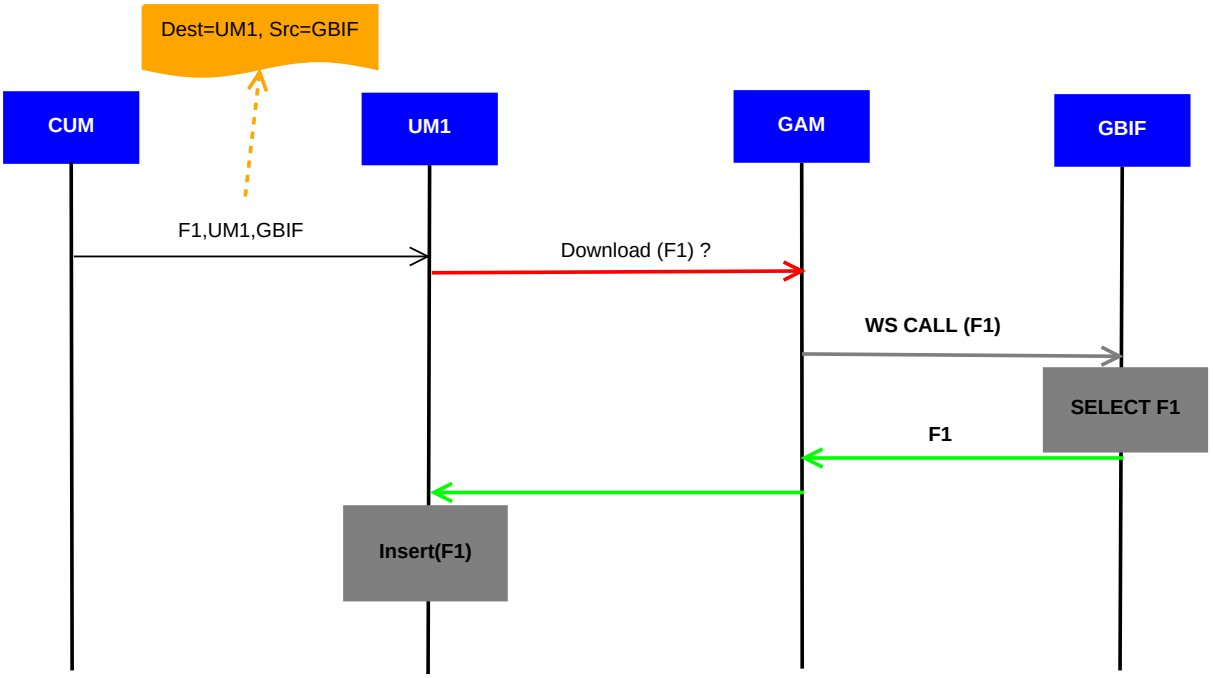

FIGURE 5.6 – Mécanisme de téléchargement de données manquantes

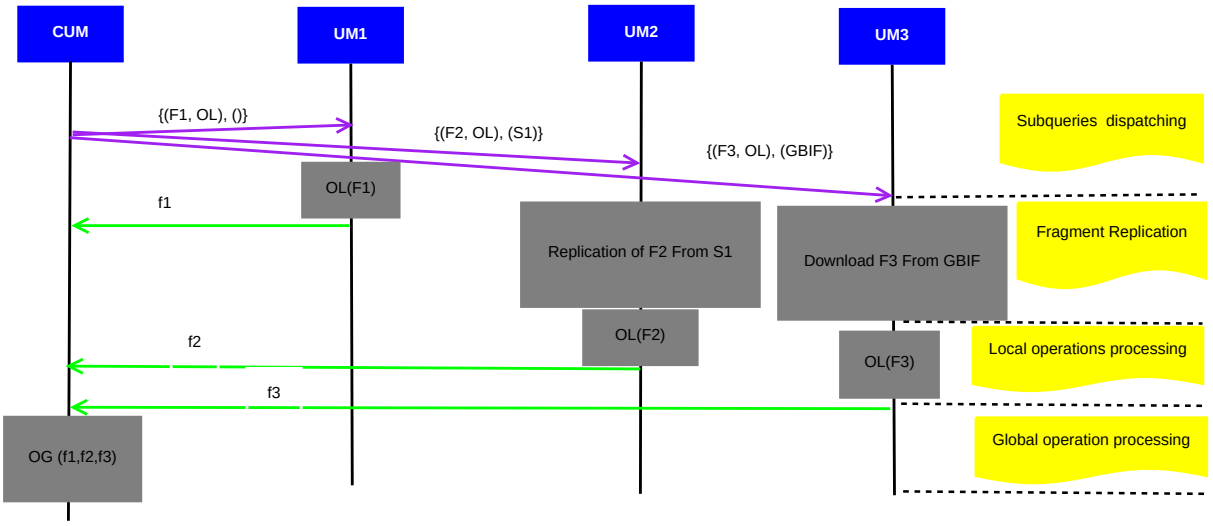

FIGURE 5.7 – Traitement réparti de requêtes

Le CUM envoie chaque sous-requête au site concerné avec les éventuelles demandes de réplication. A la réception d'une sous-requête, le UM vérifie si une réplication (ou un téléchargement à partir du portail GBIF) de fragment est nécessaire. Si c'est le cas, il procède à la réplication du fragment (ou son téléchargement) avec le processus expliqué précédemment. Lorsque toutes les données sont dans la base locale (i.e. lorsque les éventuelles réplications sont terminées), le UM envoie la sous-requête à son LM. Ce dernier traite ou met en attente (en fonction de l'occupation) la sous-requête. A la fin du traitement de la sous-requête, le LM envoie le résultat (partiel) au CUM. Ce dernier attend les résultats partiels de toutes les sous-requêtes pour traiter les opérations OG. A la fin des traitements finaux (OG), il envoie le résultat à l'utilisateur. Le traitement de la requête est illustré à travers la figure 5.7.

5.3 Exécution concurrente de plusieurs requêtes

Dans un contexte où de nombreux utilisateurs posent des requêtes, chaque site peut recevoir des requêtes et les traiter. Lorsqu'une requête arrive, le site qui coordonne les traitements OL et OG est appelé le coordinateur de la requête. Ce dernier soumet les OL aux sites concernés, en parallèle. Puis il attend les réponses de tous les OL avant d'évaluer le traitement OG. Chaque site a donc deux rôles, il peut coordonner une requête, et il peut traiter des OL pour une requête coordonnée par un autre site.

Le modèle d'exécution est conçu pour utiliser, autant que nécessaire, les ressources de calcul existantes. Sachant que chaque site dispose d'un SGBD relationnel qui exploite toutes les ressources de calcul du site pour évaluer une requête, nous faisons le choix de traiter les OL en série (file) selon leur ordre d'arrivée sur un site. Le traitement en parallèle de plusieurs OL concerne des OL évaluées par des sites différents. Lorsqu'un site a terminé le traitement d'une OL, il vérifie si une OG est prête à être traitée ; le cas échéant il traite l'OG avant de poursuivre le traitement des OL en attente, et ainsi de suite.

Un coordinateur peut demander à un site S d'évaluer un traitement OL concernant un fragment qui n'existe pas encore sur le site S. Dans ce cas, le fragment est répliqué sur S, juste avant le traitement OL. Le but de la réplication étant de décharger rapidement un site trop sollicité, l'accès à un site pour lire un fragment à répliquer est traité immédiatement, sans tenir compte des éventuelles OL en attente. Cela prévient les situations d'inter-blocage à cause d'attentes mutuelles. De plus, la création d'une nouvelle réplique ne ralentit pas significativement le site qui transmet les données. L'exemple suivant illustre l'exécution d'une requête. Nous notons L_k (respectivement G_k) l'ensemble des OL (resp. des OG) affectés au site S_k.

Exemple

Considérons les trois requêtes R_1, R_2 et R_3 portant respectivement sur les ensemble de fragments

$\{F_1, F_2\}$, $\{F_2, F_3\}$ et $\{F_1, F_3\}$ avec le schéma de placement à travers les sites S_1 et S_2 ci-après :
$S_1 : \{F_1, F_2\}$; $S_2 : \{F_3\}$
On soumet au système les trois requêtes R_1, R_2 et R_3 dans l'ordre. Notons : $OL_{i,j}$ le traitement OL de la requête R_i concernant le fragment F_j, et OG_i le traitement OG de la requête R_i.

La figure 5.8 illustre l'exécution des requêtes R_1, R_2 et R_3 à travers les sites S_1 et S_2.

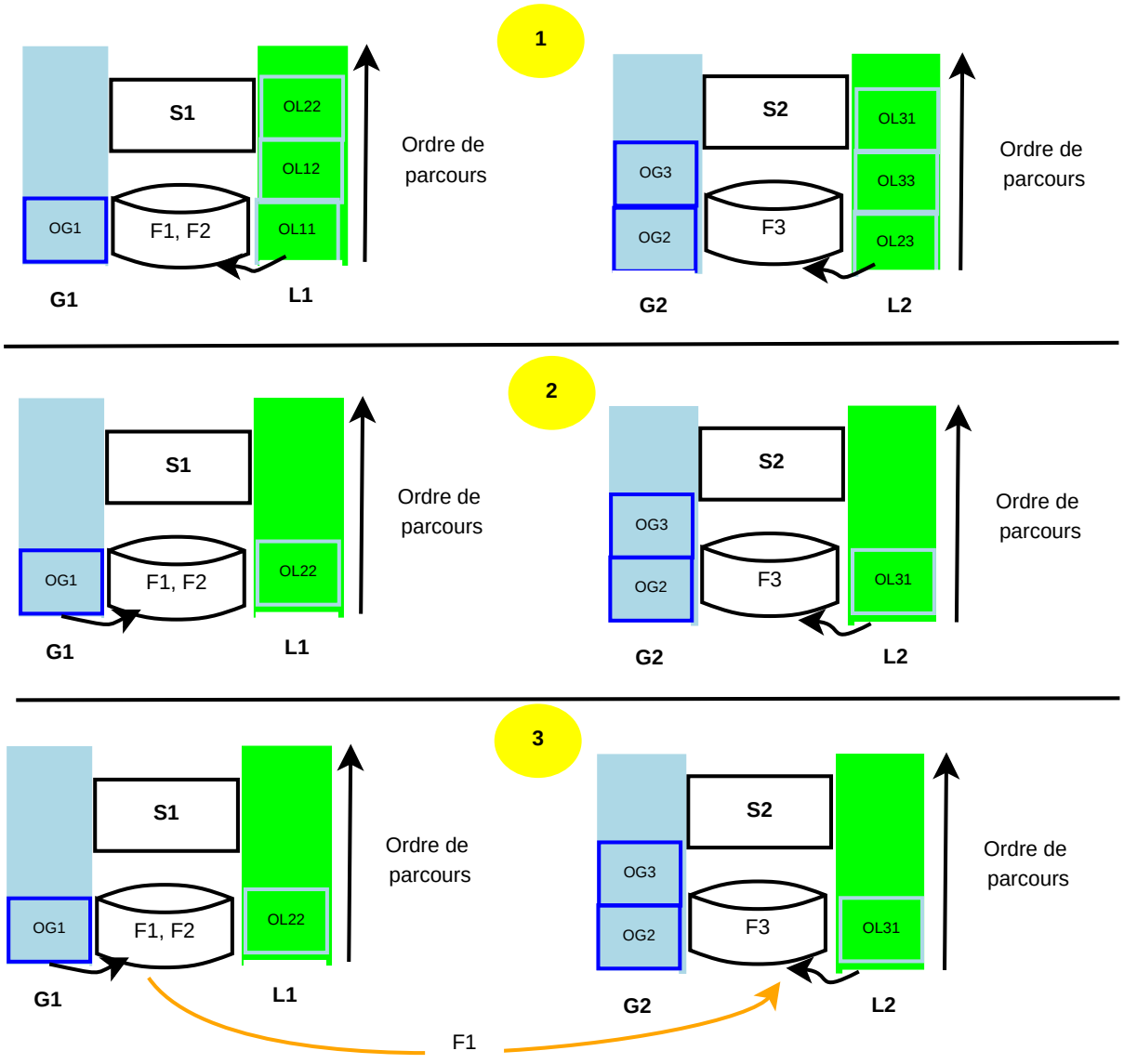

FIGURE 5.8 – Exemple d'exécution d'une requête

Nous avons $G_1 = \{OG_1\}$ en bleu sur S_1, $G_2 = \{OG_2, OG_3\}$ en bleu sur S_2 , $L_1 = \{OL_{11}, OL_{12}, OL_{22}\}$ en vert sur S_1 et $L_2 = \{OL_{23}, OL_{33}, OL_{31}\}$ en vert sur S_2. Tous les OL de R_1 sont traités au même site, S_1 qui traitera le OG de R_1. De la même façon, tous les OL de R_3 sont traités au même site, S_2 qui traitera le OG de R_3. Cependant les OL de R_2 sont réparti à travers S_1 et S_2 (étape 1). A la fin de tous les OL de R_1, le traitement du OG de la requête est démarré (étape

2). On remarque que OL_{31} portant sur le fragment F_1 est assigné au site S_2 qui ne dispose pas du fragment. De ce fait, il est nécessaire de répliquer le fragment F_1 vers S_2 pour le traitement de OL_{31} (étape 3).

5.4 Partage d'information pour l'optimisation

Pour éviter de saturer le trafic réseau, nous proposons de coordonner le partage de certaines informations entre les sites. Ces informations concernent la disponibilité et le contenu de chaque site.

Nous avons décrit la manière dont l'information sur le placement des données est partagée entre les sites dans le mécanisme d'indexation et de localisation à la section 4.4.

Pour ce qui concerne la disponibilité des sites, le catalogue global, en plus de l'information sur le contenu, garde l'information sur la disponibilité de chaque site. Pour cela, lorsqu'un site devient surchargé, il en informe au GAM qui le marque comme étant indisponible pour un nouveau traitement. De la même façon, lorsque le site redevient non-surchargé, il en informe au GAM qui le marque comme étant disponible.

Lors de l'optimisation d'une requête, le gestionnaire de requêtes demande au GAM les sites disponibles. Puis, il contacte ces sites pour obtenir les informations de leurs charges courantes. Ceci permet d'éviter les demandes de charges des sites déjà surchargés.

Seules les informations de disponibilité et de contenu des sites distants sont partagées via le catalogue global. L'intérêt de cette centralisation est de garantir la disponibilité et la cohérence des informations sur l'état de chaque site tout en réduisant le coût des communications.

5.5 Conclusion

Dans ce chapitre, nous avons présenté un modèle de requêtes d'analyse de biodiversité et les principes généraux du traitement réparti de la requête d'analyse de données de biodiversité à travers l'architecture décentralisée. Le modèle de requêtes permet d'identifier les fragments impliqués par la requête et deux types d'opérations. Il s'agit des opérations locales (qui sont appliquées sur chaque fragment) et des opérations globales (qui sont appliquées sur les résultats des premières). Puis, nous avons ensuite présenté les étapes du traitement d'une requête en détaillant le mécanisme d'analyse et les principes généraux d'optimisation de la requête. Nous avons aussi expliqué le processus de téléchargement de donnée vers les sites locaux à partir du GBIF et le processus de réplication d'un fragment d'un site local à un autre. Ensuite, nous avons expliqué le mécanisme de traitement des opérations de la requête. Enfin, nous avons montré

l'exécution concurrente de plusieurs requêtes avant d'expliquer le partage d'informations entre les sites pour l'optimisation de requêtes.

Chapitre 6

Optimisation de requêtes basée sur un coût dynamique

Le traitement de requêtes dans un environnement réparti à large échelle est assez complexe et nécessite une stratégie d'optimisation pour traiter les requêtes des utilisateurs dans des délais raisonnables. L'optimisation décide de la manière dont les opérations de la requête sont exécutées sur les données à travers les sites.

Il existe de nombreuses possibilités pour exécuter une requête. Chacune des répliques d'un fragment peut être utilisée pour traiter une requête qui porte sur le fragment. Le choix de la réplique à traiter relève de l'optimisation. De plus, dans l'objectif de traiter efficacement les requêtes, il peut être intéressant d'adapter dynamiquement le placement des données en fonction des requêtes. Le placement dynamique des données engendre des coûts de calcul, de stockage et de transfert. Le placement des nouvelles répliques peut dégrader les performances. Pour la garantie de performance, notre approche d'optimisation contrôle le placement dynamique en décidant des fragments concernés ainsi que de la source et de la destination pour une nouvelle réplique. Par ailleurs, le traitement réparti de la requête à travers plusieurs sites permettrait de réduire le temps de réponse en bénéficiant du parallélisme de certaines opérations.

Dans la suite de ce chapitre nous présentons des concepts inhérents à l'optimisation de requêtes. Puis, nous présentons notre modèle de coût dynamique. Ensuite, nous détaillons notre approche d'optimisation basée sur le coût et présentons ses avantages. Enfin, nous présentons d'autres solutions d'optimisation possibles.

6.1 Concepts et définitions

Nous nous basons sur le modèle de requêtes que nous avons présenté à la section 5.1 et les principes d'exécution concurrente de plusieurs de requêtes à la section 5.3. Nous définissons quelques paramètres de coûts inhérents au traitement d'une requête.

Definition 6.1. *Durée d'un traitement OL,* $\boldsymbol{TOL_i(F_j)}$
C'est la durée du traitement OL portant sur le fragment F_j sur le site S_i lorsque toutes les ressources (de calcul de mémoire) sont à disposition au site S_i. Ce paramètre dépend des performances du site et de la taille du fragment à traiter. A chaque traitement OL sur le site, son LM garde le temps effectué pour ce traitement en spécifiant le fragment concerné.

Definition 6.2. *Durée d'un traitement OG,* $\boldsymbol{TOG_i}$
C'est la durée d'un traitement OG portant sur l'ensemble des résultats des OL d'une requête, sur le site S_i. Puisque les tailles des résultats des OL sont relativement petites, la durée du traitement OG dépend principalement des performances du site. Chaque site calcule son TOG_i moyen après chaque exécution d'un traitement OG dans la base locale.

Definition 6.3. *Durée d'une réplication,* $\boldsymbol{TREP_{ik}(F_j)}$
C'est la durée de réplication du fragment F_j depuis le site source S_i vers le site S_k. Ce paramètre totalise le temps de lire le fragment à la source, transférer les données et insérer le fragment sur S_k. Il dépend de la taille du fragment F_j et de la capacité du lien de communication entre S_i et S_k. Lorsque le fragment existe au site S_k, alors aucune réplication n'est effectuée. De ce fait, cette durée est nulle lorsque $i = k$. Pour connaitre la valeur de $TREP_{ik}(F_j)$, le site S_k qui coordonne la réplication, mesure la délai passé depuis le début de la réplication jusqu'à la fin de l'insertion des données dans la base locale. Puis, il journalise ce délai qui pourra être utilisé pour les prochaines optimisations.

Definition 6.4. *Temps d'attente,* $\boldsymbol{TA_k}$
C'est la latence qu'un nouveau traitement aurait attendu sur le site S_k. Elle est définie par la charge courante sur le site S_k. Cette charge est la somme des durées des OL et des OG qui sont en attente sur le site. Puisqu'un OL peut nécessiter une réplication qui est coordonnée par le site lui-même, la durée de la réplication est une charge de plus induite. Ainsi nous évaluons le temps d'attente ou charge d'un site S_k à l'aide de la formule ci-dessous :

$$\mathrm{TA}_k = \sum OLn \in \mathrm{L}_k(TOL_n + \mathrm{TREP}_{jk}(\mathrm{F}_n)) + \sum OGi \in \mathrm{G}_k(TOG_i)$$

Definition 6.5. *Temps de réponse maximal,* $\boldsymbol{TR\_MAX}$
Le temps de réponse maximal, noté TR_MAX est la borne du temps de réponse à respecter pour la requête.

Definition 6.6. *Durée de transfert d'un résultat partiel,* $\boldsymbol{TP_{ki}}$
La durée de transfert, notée TP_{ki}, d'un résultat partiel issu d'un OL traité au site S_k est le temps nécessaire pour acheminer ce résultat au coordinateur S_i de la requête concernée.

Definition 6.7. *Surcharge, SS_k*
La surcharge d'un site est l'incapacité de celui-ci à effectuer au moins un traitement OL et un traitement OG de plus pour une requête tout en respectant le temps de réponse maximal.
On obtient l'information concernant la surcharge d'un site S_k grâce à la variable booléenne SS_k telle que :

$\mathrm{SS}_k \longleftarrow (\mathrm{TA}_k + \mathrm{TOL}_k \geq TR\_MAX - TOG_k)$
où TOL_k correspond à TOL moyen au site S_k.

Definition 6.8. *Plan d'exécution d'une requête*
Un plan est constitué d'un ensemble d'association entre un site et un fragment. Il est matérialisé par une liste de couples. Chaque couple étant composé d'un site et d'un ensemble de OL à traiter sur ce site. Il est complété par le coordinateur de la requête qui doit effectuer le traitement OG.

Definition 6.9. *Coût d'un plan d'exécution*
Le coût d'un plan d'exécution est le temps réponse de la requête R concernée lorsqu'elle est exécuté avec ce plan. Ce temps de réponse correspond au délai entre la réception de la requête de l'utilisation jusqu'à la fin du traitement du plan d'exécution correspondant. La formule du plan d'exécution correspond au temps de réponse T_R de la requête R. Elle est expliquée à la section 6.2.

Definition 6.10. *Candidat*
Un candidat pour une requête R est un plan d'exécution de R dont le coût T_R est tel que :
$T_R < TR\_MAX$.

6.2 Modèle de coût dynamique

Lors du traitement d'une requête, divers scénarios peuvent se présenter en fonction de la localisation des répliques à traiter : traitement local (OL et OG) de la requête sans transfert de données, traitement local avec transfert, traitement réparti avec réplication et traitement réparti sans réplication.

Nous proposons un modèle de coût généraliste qui s'adapte à tous les scénarios et qui intègre plusieurs paramètres susceptibles d'impacter le temps de réponse d'une requête.

Le temps de réponse d'une requête dépend des charges des différents sites qui sont impliqués, des charges créées par la requête sur chaque site et des éventuels coûts de transferts de résultats partiels vers le coordinateur. Nous exprimerons le temps de réponse noté T_R, en nous basant sur le modèle d'exécution concurrentes des requêtes et en utilisant les paramètres de coûts de notre approche d'optimisation. Cette estimation se fait en fonction des temps d'attente des différents

sites qui interviennent dans le traitement de la requête, des coûts des traitements OL sur ces sites et des coûts de transfert des éventuelles réplications et/ou des résultats partiels.

Puisque les traitements OL ainsi que leurs attentes à travers les sites se font en parallèle lorsque la requête est répartie, le temps nécessaire pour que les résultats de tous les OL de la requête soient disponibles au coordinateur correspond au délai qu'aura fait le dernier site à avoir envoyé son résultat au coordinateur. A ce délai s'ajoute le temps nécessaire pour le traitement OG de la requête sur les résultats de OL. Ainsi le temps de réponse T_R de la requête R est :

Definition 6.11. *Temps de réponse d'une requête, T_R*

$$T_R = max_k\ [TA_k + \sum_{OL_k \in L_k^R} (TOL_k + TREP_{jk}(F_{OL_k})) + TP_{ki}] + TOG_i$$

Où :

- S_k est un site participant au traitement de la requête (qui doit exécuter au moins un OL de la requête),
- S_i est le coordinateur de la requête qui doit exécuter le traitement OG de la requête,
- L_k^R est l'ensemble des OL de la requête R assignés au site S_k,
- TA_k est le temps d'attente des OL de la requête, assignés au site S_k,
- OL_k est un OL assigné au site S_k,
- F_{OL_k} est le fragment concerné par le traitement OL_k,
- TOL_k est le temps du traitement OL_k,
- $TREP_{jk}(F_{OL_k})$ est le temps de réplication du fragment F_{OL_k} du site S_j vers le site. D'où $TREP_{jk}(F_{OL_k}) = 0$ lorsque S_k dispose du fragment F_{OL_k},,
- TP_{ki} temps moyen de transfert du résultat partiel d'un OL de la requête du participant S_k vers le coordinateur S_i. D'où $TP_{ki} = 0$ lorsque k=i,
- TOG_i est le temps moyen d'un traitement OG au coordinateur S_i.

Ce modèle nous permet de déduire que le coût d'une requête dépend :
- des charges courantes des différents sites impliqués au traitement de la requête TA_k,
- des performances de tous les participants : TOL_k, TOG_i,
- des capacités des liens de communication entre les sites : $TREP_{jk}(F_{OL_k})$, TP_{ki},
- des propriétés des fragments invoqués dans le traitement de la requête : TOL_k, $TREP_{jk}(F_{OL_k})$
- des propriétés de la requête : TOL_k, TOG_i

Ces différents paramètres intégrés dans ce modèle visent à obtenir une meilleure efficacité pour l'optimisation de requêtes dans un environnement réparti hétérogène à large échelle. Il importe de noter que les approches d'optimisation de requêtes existantes n'intègrent pas tous ces paramètres dans leur modèle de coût (voir la section d'optimisation 2.2 de l'état de l'art).

6.3 Optimisation dynamique de requêtes

La stratégie d'optimisation de requêtes permet de trouver un plan d'exécution de la requête en fonction des paramètres qui sont pris en compte. Comme paramètres d'optimisation, nous considérons : la charge de chaque site, les localisations des répliques de chaque fragment invoqué, les capacités de calcul et de stockage de chaque site, les capacités des liens de communication entre les sites, la nature des opérations de la requête et la structure des données.

Ayant pour objectif de garantir un temps de réponse borné pour la requête, notre approche utilise les sites non-surchargés qui disposent de fragments de la requête. Elle supporte d'une part, la possibilité de répliquer les données de la requête qui se trouvent sur des sites surchargés, et d'autre part, l'adaptation du nombre de sites qui interviennent dans le traitement pour réduire les situations de blocage dues par exemple à la surcharge d'un site. En effet, lorsqu'un participant au traitement d'une requête est surchargé, le traitement final de la requête au coordinateur peut être retardé par celui-ci.

Dans la suite, nous présentons les principes de notre approche d'optimisation et proposons des algorithmes qui permettent de calculer le plan d'exécution de la requête en fonction de l'ensemble des paramètres d'optimisation considérés.

6.3.1 Principes généraux

Étant donné une requête, il s'agit de déterminer son plan d'exécution. Un plan précise les sites participants qui effectueront les traitements OL et le coordinateur qui effectuera le traitement OG.

Afin de réduire le risque d'un blocage lié à la surcharge d'un site, nous minimisons le nombre de participants, tant que le temps de réponse estimé reste inférieur à TR_MAX.

Nous déterminons, dans un premier temps, les sites qui peuvent traiter tous les OL de la requête sans réplication supplémentaire de données.

S'il n'existe pas de plan garantissant la contrainte de temps de réponse borné sans la moindre réplication supplémentaire de fragment, alors nous cherchons un plan qui garantit cette contrainte en répliquant les fragments stockés sur des sites surchargés.

S'il n'existe aucun plan garantissant la contrainte de temps de réponse borné (par exemple, cela se produit lorsque les temps de réplication sont élevés), alors nous répliquons les fragments stockés sur des sites surchargés vers le site qui a le plus de données de la requête. Cela vise à regrouper davantage les fragments invoqués ensemble tout en augmentant les possibilités de choix pour les prochaines requêtes concernant les mêmes données.

Afin de réduire le volume du transfert des résultats des traitements OL, nous désignons comme coordinateur de la requête, le site devant évaluer le plus de traitement OL de la requête.

6.3.2 Calcul de plan d'exécution sans création de nouvelle réplique

La première phase du calcul de plan d'exécution pour une requête consiste à déterminer un plan sans la nécessité de créer une nouvelle réplique. Il utilise les fragments existants pour les traitements OL. Le nombre de possibilité pour le traitement OL sur un fragment est égal au nombre de répliques du fragment à travers les sites. Le calcul de plan d'exécution considère les répliques qui satisfont la contrainte sur le temps de réponse. Parmi ces répliques, il choisit celle qui doit être utilisée pour le traitement OL concerné. Dans la suite, nous présentons ses principes et détaillons ses algorithmes.

6.3.2.1 Principes

Nous identifions les sites candidats qui satisfont à eux seuls la contrainte de temps de réponse borné sans la moindre réplication de donnée. Lorsque de tels sites existent, alors, le site le moins chargé est choisi pour traiter tous les OL de la requête.

Par contre, lorsqu'il n'existe pas de site pouvant traiter seul toutes les OL de la requête, alors nous déterminons le site qui pourrait traiter le maximum d'OL sans réplication. Pour cela, Nous déterminons les sites qui stockent des fragments de la requête et qui ne sont pas surchargés. Parmi ces derniers, nous commençons par le site qui pourrait traiter le plus grand nombre de OL tout en satisfaisant la contrainte de temps de réponse. Nous lui affectons en priorité les OL concernant les fragments les moins répliqués dont il dispose. Cet ordre d'attribution des OL permet de traiter les autres fragments (plus répliqués) dans les autres sites. Après chaque attribution d'OL à un site, nous incrémentons la charge du site concerné de la durée du traitement OL (TOL du fragment sur le même site). Les OL restants pour la requête sont affectés aux autres sites selon le même principe.

6.3.2.2 Algorithme

Cet algorithme nommé calculerPlanSansReplication (algorithme 1), est constitué des étapes successives ci-dessous où F représente tous les fragments de la requête et S l'ensemble des sites. Nous notons F^i l'ensemble des fragments de F stockés sur le site S_i

1. déterminer l'ensemble des sites non-surchargés de S (notés SNC) hébergeant au moins un fragment de la requête.
2. ordonner de façon décroissante les sites de SNC par nombre de fragments hébergés de la

requête.

3. parcourir les sites S_i de SNC (en commençant par le site qui contient plus de fragments de la requête)

3.1. ordonner chaque ensemble F^i par degré de réplication croissant

3.2. enlever de F^i les fragments de la requête qui sont déjà assignés.

3.3. vérifier que le site S_i peut traiter tous les fragments (${F_j}^i$) de la requête qu'il héberge et qui ne sont pas encore assignés (F^i)avec le respect de la contrainte de temps de réponse borné.

3.3.1. lorsque la condition est vérifiée, on assigne les traitements OL des fragments concernés au site. Une fois un fragment assigné, il n'est plus considéré pour les sites suivants.

3.3.2. dans le cas contraire, on vérifie que le site S_i peut traiter tous les fragments (F^i) de la requête qu'il héberge sauf le fragment plus répliqué (${F_{jmax}}^i$) en procédant de la même façon que précédemment.

4. lorsque tous les fragments sont assignés pour leurs traitements OL (FA = F), alors un plan d'exécution sans réplication est retourné.

5. lorsque les fragments ne sont pas tous assignés pour leurs traitements OL, alors l'algorithme retourne un résultat vide.

Fonction calculerPlanSansReplication(F : **Fragments de la requête**) : **Plan**
plan = Nouveau Plan
SNC ⟵ { $S_k \in S / \overline{SS_k}$**ET** $(F^k \neq \oslash)$}
TrierSitesDesc(SNC, F)
FA ⟵ $\oslash$
Pour chaque S_i de SNC **faire**
TrierDegreFrag(F^i)
F^i ⟵ F^i - FA
FA^i ⟵ $\oslash$
OK ⟵ FAUX
Tant que ((OK) **OU** ($F^i = \oslash$))**faire**
jmax ⟵ j / DegreRep($F_j{}^i$) = max(DegreRep(F^i))
Si ($TA_i + \sum_{F_j \in F^i} TOL_i(F_j) \leq TR\_MAX - TOG_k$)**Alors**
FA^i ⟵ F^i
OK ⟵ VRAI
Sinon
F^i ⟵ F^i - $F_{jmax}{}^i$
Fin Si
Fait
Si ($FA^i \neq \oslash$)**Alors**
Ajouter(plan, S_i, FA^i)
FA ⟵ FA $\bigcup FA^i$
Fin Si
Fin Pour
Si (FA = F) **Alors**
Retourner plan
Sinon
Retourner NULL
Fin Si
Fin

Algorithme 1 – Plan d'exécution sans création de nouvelle réplique

6.3.2.3 Exemples

Exemple 1

-Considérons les sites S_1, S_2, S_3 et S_4 avec les charges respectives suivantes
TA_1 = 1000ms, TA_2 =900 ms , TA_3 =50ms , TA_4 = 4000ms

-Considérons les fragments F_1, F_2, F_3 de la requête avec les localisations et les temps OLs suivants

$F_1 : S_1, S_2$; $F_2 : S_2$; $F_3 : S_2$

$TOL_1(F_1) = 400ms$, $TOL_1(F_2) = 600ms$, $TOL_1(F_3) = 300ms$, $TOL_1 = 400ms$

$TOL_2(F_1) = 200ms$, $TOL_2(F_2) = 300ms$, $TOL_2(F_3) = 150ms$, $TOL_2 = 200ms$

- On pose $TRMAX = 2000ms$ et $TOG_1 = TOG_2 = TOG_3 = 100ms$

1. Sites non-surchargés hébergeant au moins un fragment de la requête

S_1 ?

$TA_1 + TOL_1 ?< TR\_MAX\text{-}TOG_1$

$1000 + 400 < 2000 - 100$ **OK**

S_2 ?

$TA_2 + TOL_2 ?< TR\_MAX - TOG_2$

$900 + 200 < 2000 - 100$ **OK**

$F^1 = F_1$

$F^2 = \{F_1, F_2, F_3\}$

$SNC = \{S_1, S_2\}$

2. Ordonnons (de façon décroissante) les sites par nombre de fragments hébergés de la requête

$S_2 : 3$; $S_1 : 1$

3. Parcours des sites de SNC.

S_2 :

3.1. Ordonnons F^2 par degré de réplication

$F_1 : 2$, $F_2 : 1$, $F_3 : 1$

3.3. $TA_2 + \sum_{F_j \in F^2} TOL_2(F_j) ?< TR\_MAX - TOG_2$

$TA_2 + (TOL_2(F_1) + TOL_2(F_2) + TOL_2(F_3)) ?< TR\_MAX - TOG_2$

$900ms + (200ms + 300ms + 150ms) ? < 2000ms\text{-}100ms$

3.3.1. $1550\ ms < 1900$ **OK**

$FA^2 = \{F_1, F_2, F_3\}$

$FA = \{F_1, F_2, F_3\}$

4. $F = FA$

5. Alors le plan calculé pour les traitements OL est : **(S_2 ; {F_1, F_2, F_3})**

Exemple 2

On reprend l'exemple 1 avec une charge du site S_2 **$TA_2 = 1400$**

1. Sites non-surchargés hébergeant au moins un fragment de la requête :

S_1 ?

$TA_1 + TOL_1 \ ? < \ TRMAX\text{-}TOG_1$

$1000 + 400 < 2000 - 100$ **OK**

S_2 ?

$TA_2 + TOL_2$? < TRMAX-TOG_2

1400 + 200 < 2000 - 100 **OK**

$F^1 = F_1$

$F^2 = \{F_1, F_2, F_3\}$

SNC = $\{S_1, S_2\}$

2. Ordonnons (de façon décroissante) les sites par nombre de fragments hébergés de la requête

S_2 : 3 ; S_1 :1

3. Parcours des sites de SNC.

S_2

3.1. Ordonnons F^2 par degré de réplication

F_1 : 2, F_2 : 1, F_3 : 1

3.3. $TA_2 + \sum_{F_j \in F^2} TOL_2(F_j)$? < $TR\_MAX - TOG_2$

$TA_2 + (TOL_2(F_1) + TOL_2(F_2) + TOL_2(F_3))$? < $TR\_MAX - TOG_2$

1400ms + (200ms + 300ms + 150ms) ?< 2000ms-100ms

3.3.2. 2050 ms ≥ 1900 **!OK**

$F_{jmax}{}^2 = F_1$

$F^2 = F^2 - F_{jmax}$

$F^2 = \{F_2, F_3\}$

3.3. $TA_2 + \sum_{F_j \in F^2} TOL_2(F_j)$? < $TR\_MAX - TOG_2$

$TA_2 + (TOL_2(F_2) + TOL_2(F_3))$? < $TR\_MAX - TOG_2$

3.3.1.1400ms + (300ms + 150ms) < 2000ms-100ms **OK**

$FA^2 = \{F_2, F_3\}$

FA = $\{F_2, F_3\}$

3.1.

S_1

$F^1 = \{F_1\}$

3.3. $TA_1 + \sum_{F_j \in F^1} TOL_1(F_j)$? < $TR\_MAX - TOG_1$

$TA_1 + (TOL_1(F_1))$? < $TR\_MAX - TOG_1$

1000ms + (400ms) ? < 2000ms-100ms

3.3.2. 1400ms < 1900 **OK**

$FA^1 = \{F_1\}$

FA = $\{F_1, F_2, F_3\}$

4. F = FA

5. Alors le plan calculé pour les traitements OL est : **(S_2 ; {F_2, F_3}, S_1 ; {F_1})**

6. Le coordinateur est le site, S_2 qui a la plus grande portion de données pour la requête

Exemple 3

Reprenons l'exemple 1 avec une charge du site **S_2 TA_2 = 1600 ms**

1. Sites non-surchargés hébergeant au moins un fragment de la requête :

S_1 ?

TA_1 + TOL_1 ? < TRMAX-TOG_1

1000 + 400 < 2000 - 100 **OK**

S_2 ?

TA_2 + TOL_2 ? < TRMAX-TOG_2

1600 + 200 < 2000 - 100 **OK**

$F^1 = F_1$

$F^2 = \{F_1, F_2, F_3\}$

SNC = $\{S_1, S_2\}$

2. Ordonnons (de façon décroissante) les sites par nombre de fragments hébergés de la requête

S_2 : 3 ; S_1 :1

3. Parcours des sites de SNC.

S_2

3.1. Ordonnons F^2 par degré de réplication

F_1 : 2, F_2 : 1, F_3 : 1

3.3. $TA_2 + \sum_{F_j \in F^2} TOL_2(F_j)$? < $TR\_MAX - TOG_2$

TA_2 + ($TOL_2(F_2) + TOL_2(F_3)$) ? < $TR\_MAX - TOG_2$

1600ms + (200ms + 300ms + 150ms) ? < 2000ms-100ms

3.3.2. 2250 ms ≥ 1900 **!OK**

${F_{jmax}}^2 = F_1$

$F^2 = F^2 - F_{jmax}$

$F^2 = \{F_2, F_3\}$

3.3. $TA_2 + \sum_{F_j \in F^2} TOL_2(F_j)$? < $TR\_MAX - TOG_2$

TA_2 + ($TOL_2(F_2) + TOL_2(F_3)$) ? < $TR\_MAX - TOG_2$

3.3.3.1600ms + (300ms + 150ms) ≥ $2000ms - 100ms$ **!OK**

${F_{jmax}}^2 = F_2$

$F^2 = F^2 - F_{jmax}$

$F^2 = \{F_3\}$

3.3. $TA_2 + \sum_{F_j \in F^2} TOL_2(F_j)$? < $TR\_MAX - TOG_2$

TA_2 + ($TOL_2(F_3)$) ? < $TR\_MAX - TOG_2$

3.3.1.1600ms + (150ms) < 2000ms-100ms **OK**

$FA^2 = \{F_3\}$

FA = $\{F_3\}$

S_1

3.1. Ordonnons F^1 par degré de réplication

F_1 : 2

3.3. $TA_1 + \sum_{F_j \in F^1} TOL_1(F_j)$? < $TR\_MAX - TOG_1$

$\mathrm{TA}_1 + (\mathrm{TOL}_1(F_1))$? $< TR\_MAX - TOG_1$
3.3.1.1000ms + (400ms) < 2000ms-100ms **OK**
$FA^1 = \{\mathrm{F}_1\}$
$\mathrm{FA} = \mathrm{FA} \bigcup FA^1$
$\mathrm{FA} = \{\mathrm{F}_3, \mathrm{F}_1\}$
5. $FA \neq F$

Dans ce cas, on applique l'algorithme de calcul de plan avec la réplication de données (voir plus loin).

6.3.3 Calcul de plan d'exécution avec création de nouvelle réplique

Cette section détaille le cas où il n'existe aucun plan pour traiter la requête avec le respect du temps de réponse borné, en utilisant les fragments existants. Il est nécessaire de répliquer davantage les données. Dans ce cas, nous proposons de répliquer des fragments stockés sur des sites surchargés vers des sites disponibles, afin de satisfaire la contrainte de temps de réponse borné.

La réplication a pour effet d'élargir les possibilités de plan d'exécution pour les requêtes ultérieures. Nous intégrons la réplication dans le processus d'optimisation de la requête. Nous prenons en compte le coût de la réplication pour déterminer le site sur lequel nous ajoutons une réplique.

6.3.3.1 Principes

Nous n'énumérons pas exhaustivement toutes les possibilités de répliquer chaque fragment sur chaque site car le nombre de possibilités est trop élevé en général (i.e. égal à S^F pour S sites et F fragments). Nous optons pour une approche de type *greedy* reposant sur les principes suivants :

Nous utilisons autant que possible les fragments existants. Nous accédons en priorité aux sites stockant le plus grand nombre de fragments de la requête. Pour un site, nous accédons en priorité aux fragments qui sont déjà les plus répliqués. Cette première étape permet d'attribuer un site à une partie des fragments de la requête. Nous décidons de répliquer les fragments restants pour lesquels aucun site n'a été attribué. Par conséquent, les fragments les moins répliqués ont plus de chance d'être répliqués. Cela tend à répartir l'effet de la réplication sur davantage de fragments. Ce qui convient bien à notre contexte fluctuant où tout fragment peut devenir très sollicité pendant un certain temps.

La gestion des décisions de la réplication dans le processus d'optimisation de la requête engendre de nouveaux défis. Ces derniers concernent le coût de la réplication et le choix du site destinataire pour accueillir une nouvelle réplique. Après avoir présenté notre approche pour relever ces défis,

nous présentons nos algorithmes pour calculer le plan d'exécution d'une requête avec la création de nouvelle réplique.

Coût de création d'une réplique

La création d'une nouvelle réplique engendre un coût de stockage, un coût de transfert et un coût de calcul pour l'insertion dans la base destinataire. Notre approche crée une nouvelle réplique lorsque la requête courante ne peut plus être satisfaite dans des délais impartis. C'est à dire lorsque toutes les répliques du fragment concerné sont surchargées. Ceci favorise une meilleure disponibilité que si la réplication était indépendante de l'optimisation de requêtes.

Choix de site pour une nouvelle réplique

Le choix du site qui doit abriter une nouvelle réplique est important pour les performances du système. Dans notre approche, ce choix est guidé par la requête courante qui est à l'origine de la réplication. D'une part, le site choisi doit participer à la garantie du temps de réponse borné de la requête lorsque cela est possible. D'autre part, il doit participer à la réduction du temps de réponse des prochaines requêtes qui invoquent les mêmes fragments que la requête en cours d'optimisation.

Pour cela, nous choisissons dans un premier temps, de placer la nouvelle réplique sur un site qui pourrait garantir un temps de réponse borné en tenant compte des charges des sites et du coût de la réplication. Lorsque cela est possible, parmi les sites disponibles, nous choisissons le site qui dispose de plus de données de la requête s'il existe. Sinon le moins chargé est choisi. Dans le cas contraire, le fragment est répliqué vers le site qui a la plus grande portion de données de la requête.

Il importe de noter que notre solution permet aussi de traiter une grande portion de la base, car il est possible de répartir les accès aux nombreux fragments, si cela s'avère nécessaire, sur des de nombreuses machines. En outre, lorsque le site désigné n'a plus d'espace pour accueillir la nouvelle réplique, la stratégie de remplacement est appliquée immédiatement.

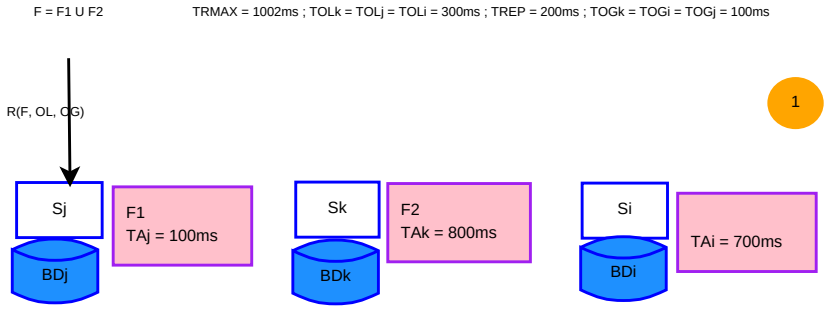

FIGURE 6.1 – Etat des sites et des paramètres à la réception d'une nouvelle requête

Les figures 6.1, 6.2, 6.3 et 6.4 montrent un exemple de choix de site pour accueillir une nouvelle réplique. Dans cet exemple le site qui dispose d'un fragment de la requête est surchargé. Le scénario 1 (figure 6.1) montre qu'une réplication s'impose pour le respect de la contrainte de temps de réponse borné. Le choix du site devant accueillir la nouvelle réplique se fait entre un site moins chargé disposant d'un autre fragment de la requête et un autre site surchargé du système. Les scénarios 2 et 3 (figures 6.3 et 6.4) correspondent chacun à un choix de site pour la nouvelle réplique. Par la suite, nous avons évalué le temps de réponse de chaque scénario par rapport au respect de la borne du temps de réponse. Notre approche correspond au scénario 2 (figure 6.3) qui choisit le site Sj comme destinataire de la nouvelle réplique et qui garantit le respect de la borne du temps de réponse.

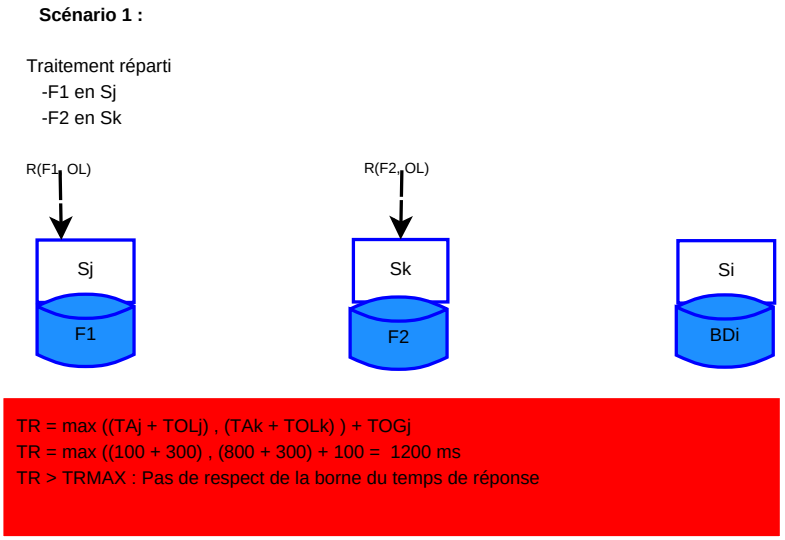

FIGURE 6.2 – Traitement réparti de la requête

6.3.3.2 Algorithmes

Les étapes successives de l'algorithme de calcul de plan avec la possibilité de réplication de données sont les suivantes :
Nous notons F^i l'ensemble des fragments de la requête stockés sur le site S_i
1. déterminer les sites non-surchargés (notés SNC) hébergeant au moins un fragment de la requête. 2. ordonner de façon décroissante les sites par nombre de fragments hébergés de la requête.
3. parcourir les sites S_i de SNC (en commençant par le site qui contient plus de fragments de la requête). 3.1. ordonner de façon croissante chaque ensemble de fragments F^i du site S_i par degré de réplication.
3.2. enlever de F^i les fragments de la requête qui sont déjà assignés.
3.3. vérifier que le site S_i pourrait traiter tous les fragments ($F_j{}^i$) de la requête qu'il héberge et

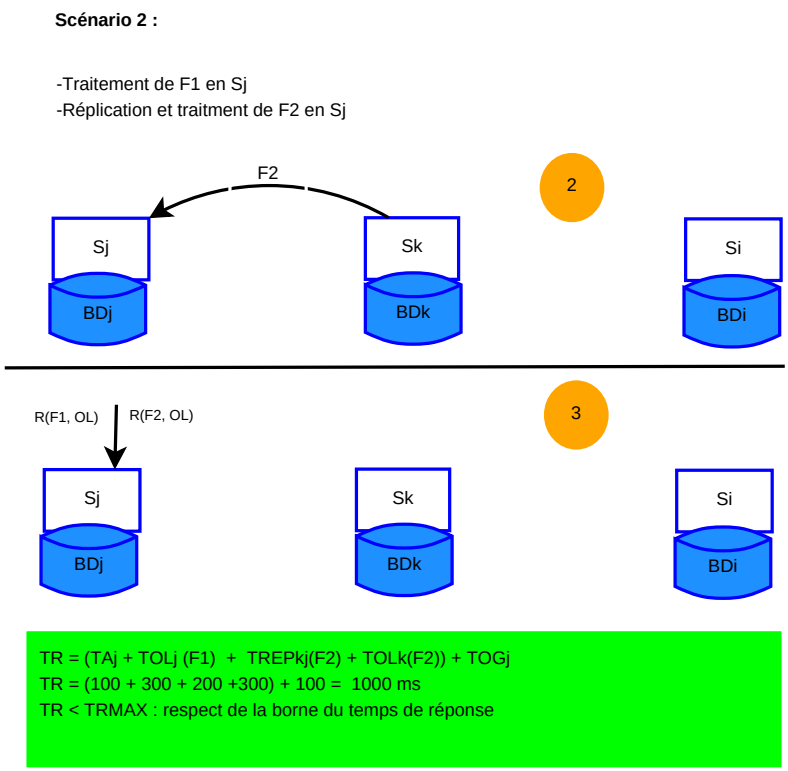

FIGURE 6.3 – Réplication avec la garantie de la borne du temps de réponse

qui ne sont pas encore assignés (F^i)avec le respect de la contrainte de temps de réponse borné.
3.3.1. lorsque la condition est vérifiée, on assigne les traitements OL des fragments concernés au site. Une fois un fragment assigné, il n'est plus considéré pour les sites suivants.
3.3.2. dans le cas contraire, on vérifie que le site S_i pourrait traiter tous les fragments (F^i) de la requête qu'il héberge sauf le fragment moins répliqué (${F_{jmin}}^i$) en procédant de la même façon que précédemment.
4. après avoir parcouru les sites ordonnés et que tous les fragments de la requête ne sont pas assignés, alors :
4.1. incrémente la charge de chaque site qui était non-surchargé des coûts des OL qu'on lui a déjà assignés pour la requête.
4.2. calculer tous les sites du système non-surchargés SNCa.
4.3. ordonner de façon décroissante les sites SNCa par nombre de fragments déjà assignés de la requête, puis par charge courante.
4.4. parcourir les sites ordonné SNCa en commençant le premier site.
4.4.1. déterminer FNA les fragments de la requête qui ne sont pas encore assignés. On a FNA = F - FA. Si FNA est vide passer, alors arrêter.
4.4.2. 4.4.2.1. ordonner de façon croissante les fragments FNA par degré de réplication.
4.4.2.2. vérifier que le site courant S_i pourrait traiter tous les fragments restants (FNA) de la requête lorsqu'ils sont répliqués dans la base locale avec le respect de la contrainte de temps de

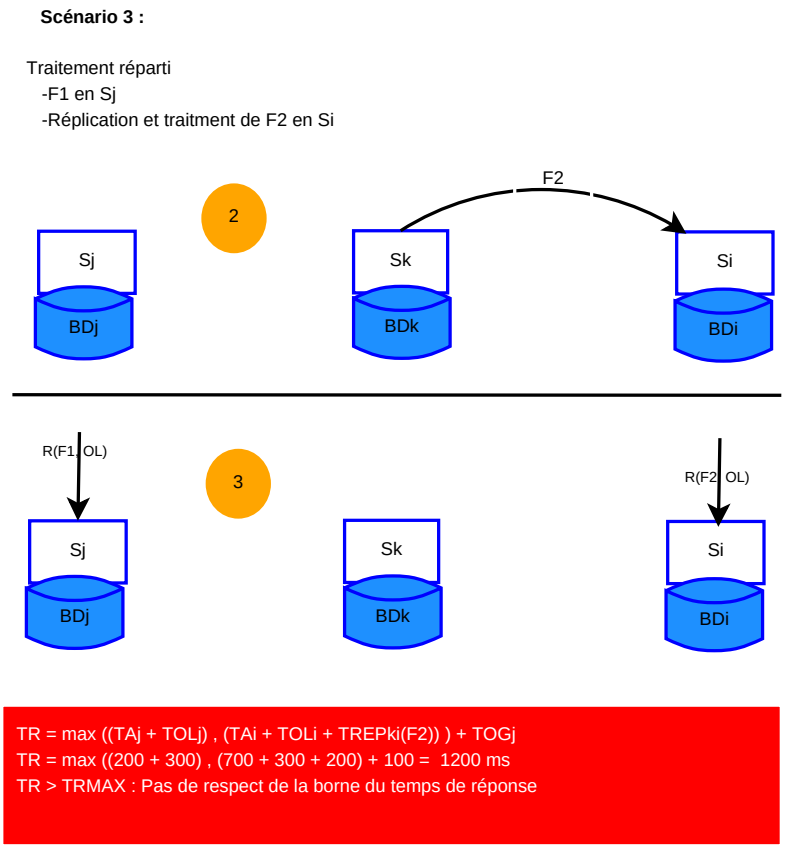

FIGURE 6.4 – Réplication sans la garantie de la borne du temps de réponse

réponse borné.

4.4.2.2.1. lorsque la condition est vérifiée, on assigne les traitements OL des fragments concernés au site. Une fois un fragment assigné, il n'est plus considéré pour les sites suivants.

4.4.2.2.2. dans le cas contraire, on vérifie que le site S_i pourrait traiter tous les fragments restants (FNA) de la requête lorsqu'ils sont répliqués dans la base locale sauf le fragment moins répliqué FNA_{jmin}en procédant de la même façon que précédemment (4.4.3).

5. après avoir parcouru les sites ordonnés et que tous les fragments de la requête ne sont pas tous assignés, alors :

5.1. déterminer le site S_{kmax} dont on a assigné le plus de OL de la requête.

5.2. Si S_{kmax} existe, alors assigner les OL (fragments) restants FNA de la requête au site S_{kmax}.

5.3. Sinon, assigne les OL (fragments) restants FNA de la requête au premier site de SNCa.

L'algorithme de calcul de plan d'exécution avec la réplication de fragment (algorithme 2)peut être décomposé en trois phases successives :

-attribution de OL sans réplication de fragment en respectant la contrainte de temps de réponse borné : de 1. à 3 (algorithme 3).

-attribution de OL avec réplication de fragment en respectant la contrainte de temps de réponse borné : 4 (algorithme 4).

-attribution de OL avec réplication de fragment sans respecter la contrainte de temps de réponse borné : 5 algorithme 5).

```
Fonction calculerPlanAvecReplication(F : Fragments de la requete) : Plan
    plan : Nouveau Plan
    FA ⟵ ⊘
    assignerTraitementSansReplicationAvecSeuil(F, FA, plan)
    assignerTraitementAvecReplicationAvecSeuil(F, FA, plan)
    assignerTraitementAvecReplicationSansSeuil(F, FA, plan)
    Retourner plan
Fin
```

Algorithme 2 – Plan d'exécution avec création de nouvelle réplique

Les algorithmes ci-dessous correspondent aux trois phases du calcul de plan d'exécution des traitements OL d'une requête.

Procédure assignerTraitementSansReplicationAvecSeuil(F : **Fragments de la requete**, FA : **Fragments assignés**, plan : **Plan**)

SNC $\longleftarrow$ { S_k / $\overline{SS_k}$**ET** $F^k \neq \oslash$}
TrierSitesDesc(SNC, F)
FA = $\oslash$
Pour chaque S_i de SNC **faire**
  TrierDegreFrag(F^i)
  $F^i \longleftarrow F^i$ - FA
  $FA^i \longleftarrow \oslash$
  OK $\longleftarrow$ FAUX
  Tant que $((\overline{OK})$**OU** $(F^i = \oslash))$**faire**
    Si $(TA_i + \sum_{F_j \in F^i} TOL_i(F_j) \leq TR\_MAX - TOG_i)$**Alors**
      $FA^i \longleftarrow F^i$
      OK $\longleftarrow$ VRAI
    Sinon
      jmin $\longleftarrow$ j / DegreRep($F_j{}^i$) = min(DegreRep(F^i))
      $F^i \longleftarrow F^i$ - $F_{jmin}{}^i$
    Fin Si
  Fait
  Si $(FA^i \neq \oslash)$**Alors**
    Ajouter(plan, S_i, FA^i)
    FA $\longleftarrow$ FA $\bigcup FA^i$
  Fin Si
Fin Pour

Fin

Algorithme 3 – Assignation de traitement sans réplication et avec contrainte

Procédure assignerTraitementAvecReplicationAvecSeuil(F : **Fragments de la requete**, FA : **Fragments assignés**, plan : **Plan**)

Pour chaque S_i de SNC **faire**

TA$_i \longleftarrow$ TA$_i + \sum_{F_j \in FA^i} TOL_i(F_j)$

Fin Pour

SNCa $\longleftarrow$ { S$_k \in S / \overline{SS_k}$}

TrierSitesAssigneDesc(SNCa, FA)

Pour chaque S_i de SNCa **faire**

FNA $\longleftarrow$ F - FA

Si (FNA $\neq \oslash$)**Alors**

OK $\longleftarrow$ FAUX

Tant que ($(\overline{OK})$**ET** $(FNA \neq \oslash)$)**faire**

cout $\longleftarrow$ TA$_i + \sum_{FNA_j \in FNA}(TOL_i(FNA_j) + TREP_{ik}$(FNA$_j$))

Si (cout $\leq TR\_MAX - TOG_i$)**Alors**

FA$^i \longleftarrow$ FA$^i \bigcup FNA$

FA $\longleftarrow$ FA $\bigcup FA^i$ OK $\longleftarrow$ VRAI

Sinon

jmin $\longleftarrow$ j / DegreRep(FNA$_j$) = min(DegreRep(FNA))

FNA $\longleftarrow$ FNA - FNA$_{jmin}$

Fin Si

Fait

Si (FA$^i \neq \oslash$)**Alors**

Ajouter(plan, S$_i$, FA^i)

FA $\longleftarrow$ FA $\bigcup FA^i$

Fin Si

Fin Si

Fin Pour

Fin

Algorithme 4 – Assignation de traitement avec réplication et avec contrainte

Procédure assignerTraitementAvecReplicationSansSeuil(F : **Fragments de la requete**, FA : **Fragments assignés**, plan : **Plan**)

TrierSitesAssigneDesc(SNCa, FA)

S_{kmax} ⟵ SNC(0)

FNA ⟵ F- FA

FA^{kmax} ⟵ $FA^{kmax} \bigcup FNA$

Ajouter(plan, S_{kmax}, FA^{kmax})

Fin

Algorithme 5 – Assignation de traitement avec réplication et sans contrainte

6.3.3.3 Exemples

Exemple 4

S_2 TA_2 = 1600 ms

Lorsqu'on constate qu'on ne peut pas obtenir un plan d'exécution qui respecte un temps de réponse borné sans réplication de données (voir exemple3), alors on applique l'algorithme de calcul de plan avec la possibilité de réplication de données

Rappelons : TA_1 = 1000ms, S_2 TA_2 = 1600 ms , TA_3 =50ms , TA_4 = 4000ms

F_1 : S_1,S_2 ; F_2 : S_2 ; F_3 : S_2

$TOL_1(F_1)$ = 400ms, $TOL_1(F_2)$ = 600ms, $TOL_1(F_3)$ = 300ms, TOL_1 = 400ms

$TOL_2(F_1)$ = 200ms, $TOL_2(F_2)$ = 300ms, $TOL_2(F_3)$ = 150ms, TOL_2 = 200ms

$TOL_3(F_1)$ = 600ms, $TOL_3(F_2)$ = 900ms, $TOL_3(F_3)$ = 500ms, TOL_3 = 700ms

$TOL_4(F_1)$ = 500ms, $TOL_4(F_2)$ = 750ms, $TOL_4(F_3)$ = 375ms, TOL_4 = 550ms

$TREP_{21}(F_3)$ = 400ms, $TREP_{23}(F_3)$ = 400ms $TREP_{21}(F_2)$ =600ms, $TREP_{23}(F_2)$=600ms

TRMAX = 2000ms et TOG_1 = TOG_2 = TOG_3 = 100ms

1. Sites non-surchargés : S_1, S_2, S_3

SNC = {S_1, S_2, S_3} 2. ordonnons de façon décroissante les sites de SNC par nombre de fragments hébergés de la requête

S_2 : 2, S_1 : 1, S_3 : 0

3. Parcours des sites de SNC.

S_2

3.1. Ordonnons F^2 par degré de réplication

F_1 : 2, F_2 : 1, F_3 : 1

3.3. $TA_2 + \sum_{F_j \in F^2} TOL_2(F_j)$? $< TR\_MAX - TOG_2$

$TA_2 + (TOL_2(F_1) + TOL_2(F_2) + TOL_2(F_3))\ ? < TR\_MAX - TOG_2$

1600ms + (200 + 300 + 150) ? < 2000ms-100ms

3.3.2. 2250ms ≥ 1900 **!OK**

3.3.2.1. $F^2 = F^2$ - F_3

$F^2 = \{F_1, F_2\}$

3.3. $TA_2 + \sum_{F_j \in F^2} TOL_2(F_j)\ ? < TR\_MAX - TOG_2$

$TA_2 + (TOL_2(F_1) + TOL_2(F_2))\ ? < TR\_MAX - TOG_2$

1600ms + (200 + 300) ? < 2000ms - 100ms

3.3.2. 2100ms ≥ 1900 **!OK**

3.3.2.1. $F^2 = F^2$ - F_2

$F^2 = \{F_1\}$

3.3. $TA_2 + \sum_{F_j \in F^2} TOL_2(F_j)\ ? < TR\_MAX - TOG_2$

$TA_2 + (TOL_2(F_1)\ ? < TR\_MAX - TOG_2$

3.3.1.1600ms + (200) < 2000ms-100ms **OK**

$FA^2 = \{F_1\}$

$FA = \{F_1\}$

S_1

$F^1 = \oslash$

S_3

$F^3 = \oslash$

4.

4.1.TA_2 = TA_2 + 200ms = 1800 ms

4.2. SNCa = ?

S_2 ?

$TA_2 + TOL_2\ ? < TR\_MAX - TOG_2$ **!OK**

$SNCa = \{S_1, S_3\}$

4.4. Parcours des sites de SNCa

S_1

4.4.1. FNA ={F_2, F_3 }

4.4.3. $TA_1 + \sum_{F_j \in FNA}(TOL_1(F_j) + TREP_{21}(F_j))\ ? < TR\_MAX - TOG_1$

1000ms + ((400ms + 600ms)+ (300ms + 600ms)) ? < 2000ms - 100ms **!OK**

4.4.3.2.$FNA_{jmin} = F_2$

4.4.3. $TA_1 + \sum_{F_j \in (FNA - FNAjmin)} (TOL_1(F_j) + TREP_{21}(F_j))\ ? < TR\_MAX - TOG_1$

1000ms + ((300ms + 400ms)) ? < 2000ms - 100ms **OK**

$FA^1 = \{F_3\}$

FA =FA $\bigcup FA^1 = \{F_1, F_3\}$

S_3

4.4.1. FNA ={F_2}

4.4.3. $TA_3 + \sum_{F_j \in FNA}(TOL_3(F_j) + TREP_{23}(F_j))\ ? < TR\_MAX - TOG_3$

50ms + (900ms + 600ms) ? < 2000ms - 100ms **OK**
$FA^3 = \{F_2\}$
FA =FA $\bigcup FA^3 = \{F_1, F_3, F_2\}$
4. F = FA
5. Alors le plan calculé pour les traitements OL est : **(S_2 ; {F_1}, S_1 ; {F_3}, S_3 ; {F_2})**
6. Le coordinateur est le site qui a la plus grande portion de données pour la requête. Puisque tous les participants ont le même nombre de OL, alors n'importe quel participant peut être le coordinateur de la requête.

6.4 Avantages de notre approche

Notre approche d'optimisation offre plusieurs avantages dans le fonctionnement global du système.

6.4.1 Garantie de temps de réponse borné pour une requête fréquente

L'approche utilise toutes les ressources (calcul et données) possibles permettant de garantir un temps de réponse borné. Si nécessaire, la requête est décomposée pour accéder en parallèle aux fragments de la requête. L'approche favorise l'accès à des données stockées sur des sites non surchargés. Cela permet de répartir les demandes pour des données populaires (qui seraient davantage répliquées) entre toutes leurs répliques. Notre approche aboutit à garantir le temps de réponse d'un plus grand nombre de requêtes accédant à des fragments populaires (voir la section 7.4 pour l'étude comparative)

6.4.2 Adaptation dynamique du schéma de placement des données

Notre approche évite la surcharge d'un site, en détectant les fragments de la requête qui se trouvent sur des sites surchargés et en les répliquant vers d'autres sites moins chargés. Ceci augmente les possibilités de choix pour les requêtes ultérieures qui invoqueront ces fragments. Le placement d'une nouvelle réplique permet de regrouper progressivement les données d'une même requête. Cela permet de traiter les requêtes impliquant les mêmes données sur un même site et ainsi réduire les surcoûts dus au traitement réparti de requêtes. Cela réduit aussi les situations de blocage inhérentes au traitement réparti (la probabilité de panne augmentant avec le nombre de sites).

6.5 Solutions comparatives pour l'optimisation des requêtes

Nous présentons les principes de quelques approches possibles qui ont pour objectif de réduire le temps de réponse des requêtes en fonction de certains paramètres qui l'affectent. Ces approches heuristiques sont plus simples que l'approche, basée sur un modèle de coût, présentée ci-dessus.

6.5.1 MiniTrans : Minimiser les transferts

Consciente de l'impact des coûts de transfert sur le temps de réponse, cette approche vise à minimiser les transferts (d'où l'appellation MiniTrans). Pour cela, elle choisit le site qui contient la plus grande partie des données impliquées pour y traiter la requête. Ainsi, toutes les données qui n'y sont pas présentes auparavant seront rapatriées. L'avantage de cette approche est que les données impliquées dans les requêtes seront davantage regroupées. De ce fait, les requêtes suivantes nécessiteront de moins en moins de transfert. Ce qui permet de réduire le temps de réponse lorsque le site qui contient toutes les données d'une requête n'est pas trop chargé.

Cependant, cette approche présente quelques limites. D'abord, elle a tendance à rapatrier les données manquantes des requêtes vers les mêmes sites qui devront davantage insérer de nouvelles données dans leur base. Puisque la capacité de stockage d'un site est limitée, elle est rapidement atteinte. Ce qui ne garantit pas un passage à l'échelle des données et des traitements. En outre, les requêtes des utilisateurs sont traitées sur ce petit nombre de sites au moment où d'autres ne sont presque pas utilisés. Ce qui n'équilibre pas la charge entre les sites (voir la section sur l'évaluation des performances 7.4) et n'optimise pas l'utilisation des ressources disponibles. Ce problème est résolu par notre approche qui n'assigne pas un nouveau traitement à un site qui est déjà surchargé. Nous illustrons le fonctionnement et les limites de cette approche à travers la figure 6.5.

6.5.2 EquiLoad : Traiter la requête sur le site le moins chargé

Puisque l'approche qui minimise les transferts n'équilibre pas la charge à travers les sites, une approche qui garantit cette fonctionnalité serait de traiter la requête au site le moins chargé en y transférant les données manquantes. L'avantage de cette approche est qu'il utilise tous les sites pour traiter les requêtes des utilisateurs. De ce fait, cette approche que nous appelons EquiLoad, optimise l'utilisation des ressources de calcul en répartissant les traitements à travers les sites (d'où son nom EquiLoad pour équilibrer la charge) . Cependant, le site le moins chargé peut ne pas disposer de données de la requête. Dans ce cas, ces données y sont transférées.

Une nouvelle requête peut nécessiter que les mêmes données soient transférées vers un autre site. On peut noter qu'un des points faibles de cette approche est qu'elle génère beaucoup de

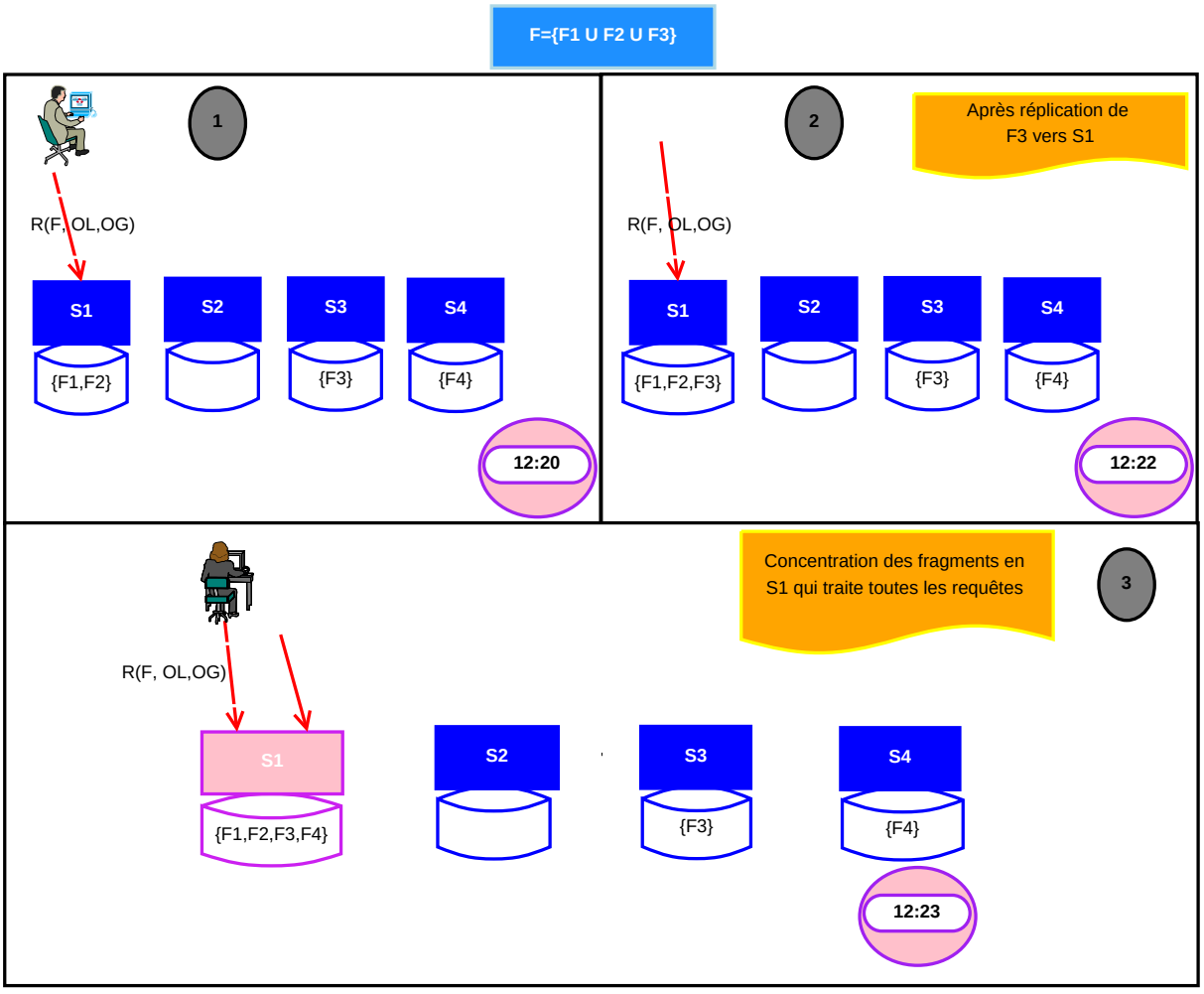

FIGURE 6.5 – Fonctionnement et limites de l'approche MiniTrans

trafic réseau. On peut supposer qu'à partir d'un certain temps toutes les données nécessaires au traitement de la requête se trouvent déjà sur le site le moins chargé puisque le système aboutit à une situation où chaque site contient le maximum de données qu'il peut stocker. Cette situation tend à remplacer continuellement le contenu d'un site. La majeure partie des ressources est utilisée pour les transferts et non pour traiter les requêtes. Pour résoudre le problème lié à la capacité de stockage limitée d'un, nous avons introduit une stratégie de remplacement qui garantit la disponibilité des données tout en préservant l'espace de stockage de chaque site. La borne du temps de réponse nous permet d'éviter les réplications inutiles effectuées par EquiLoad.

6.5.3 DisNoRep : Traitement réparti de la requête à travers les sites qui contiennent les données

Cette approche exploite les avantages du traitement réparti en découpant la requête de l'utilisateur en sous-requêtes (composées des opérations OL) en fonction de la localisation des données

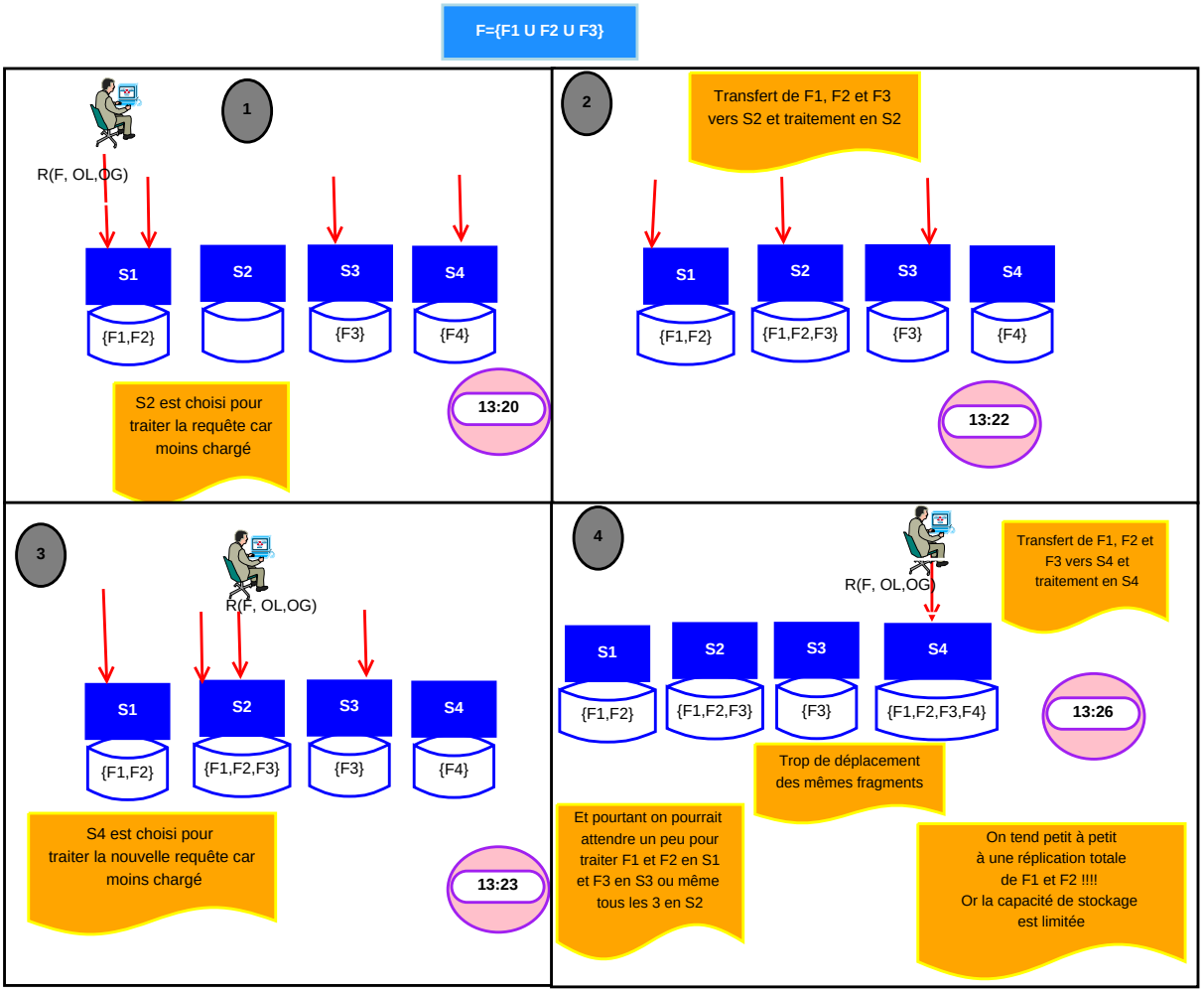

FIGURE 6.6 – Fonctionnement et limites de l'approche EquiLoad

impliquées. Avec DisNoRep 6.7 pour traitement distribué sans réplication de données, chaque sous-requête est envoyée et traitée au site hébergeant les données concernées.

Les résultats partiels des sous-requêtes sont envoyés vers un site coordinateur de la requête pour effectuer le traitement OG. Les données manquantes du système décentralisé (disponibles uniquement dans la base du GBIF), sont placées sur le site qui dispose de plus de données de la requête lorsque celui-ci existe. Sinon elles sont placées sur le site le moins chargé. Le traitement parallèle des sous-requêtes peut jouer un rôle important pour la réduction du temps de réponse puisque ces sous-requêtes sont composées des traitements OL plus coûteux. Le problème identifié avec cette approche est que le traitement OG peut être bloqué parce que le coordinateur attend un site distant qui n'a pas encore envoyé son résultat partiel. Ce retard peut être dû à une surcharge du site. Pour éviter les latences dues à une ressource critique qui est trop sollicitée, notre approche réplique les fragments de la requête qui se trouvent sur des sites surchargés.

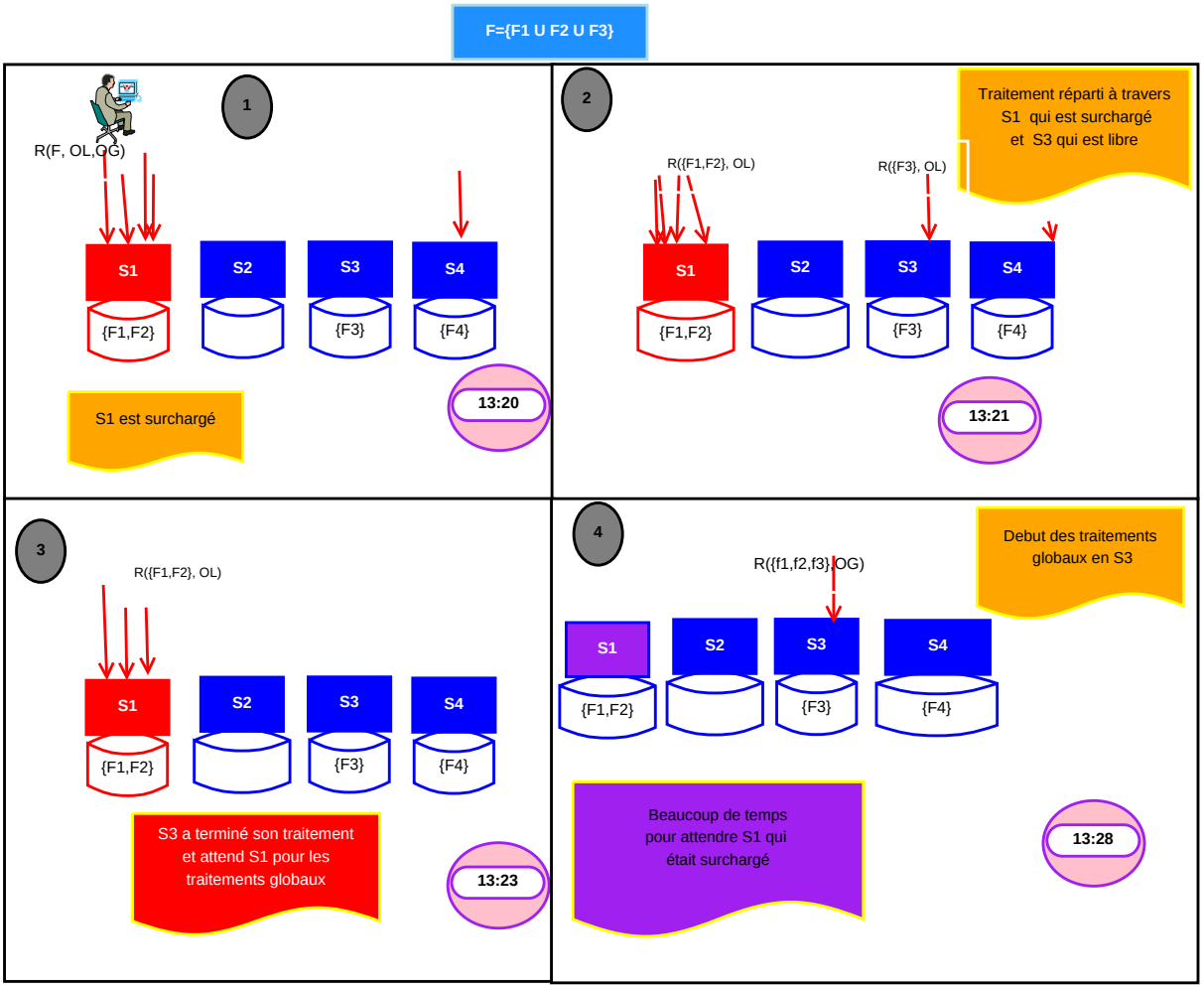

FIGURE 6.7 – Fonctionnement et limites de l'approche DisNoRep

6.5.4 Synthèse

Dans cette sous-section, nous récapitulons les propriétés des approches comparatives que nous comparons avec notre approche. Ces propriétés concernent la réduction progressive des transferts, la réduction des risques de surcharge d'un site, le respect d'un temps de réponse borné, le schéma de placement dynamique des données et la garantie du passage à l'échelle des données et des requêtes. Nous présentons dans le tableau 6.1 le comportement de chaque approche par rapport à chacune de ces propriétés.

Nous constatons que seule notre approche garantit un temps de réponse borné et le passage à l'échelle des données et des traitements. Ce qui joue important pour la garantie des performances du système pour la gestion des données de biodiversité.

Solution	réduire les transferts	éviter la surcharge	temps de réponse borné	schéma de placement dynamique	passage à l'échelle
MiniTrans	**Oui**	**Non**	**Non**	**Oui**	**Non**
EquiLoad	**Non**	**Oui**	**Non**	**Oui**	**Non**
DisNoRep	**Oui**	**Non**	**Non**	**Non**	**Non**
Notre Approche	**Oui**	**Oui**	**Oui**	**Oui**	**Oui**

TABLE 6.1 – Comparaison des approches

6.6 Conclusion

Dans ce chapitre, nous avons détaillé plusieurs approches d'optimisation de requêtes. D'abord, nous avons présenté les principes de notre approche d'optimisation et avons détaillé les algorithmes de calcul du plan d'exécution. Nous avons expliqué les avantages de notre approche. Nous avons présenté les principes et les limites de quelques approches heuristiques simples pour optimiser le traitement de requêtes dans un environnement réparti. Ayant comme objectif de garantir un temps de réponse borné pour les requêtes, notre approche d'optimisation, évite qu'un site soit surchargé en supportant le traitement réparti de la requête bien qu'il tende à minimiser le nombre de participants.

Lorsque les sites qui disposent des données de la requête ne peuvent pas garantir un temps de réponse borné pour la requête, alors notre approche d'optimisation adapte le placement des données en répliquant les fragments de la requête qui se trouvent sur des sites surchargés vers des sites qui pourront satisfaire la contrainte sur le temps de réponse pour la requête courante ou pour les prochaines requêtes. De ce fait, elle garantit un temps de réponse borné pour les requêtes qui invoquent des fragments populaires. Le modèle de coût dynamique de notre approche d'optimisation s'adapte bien au contexte hétérogène et dynamique des environnements répartis à large échelle.

Chapitre 7

Validation expérimentale

Dans ce chapitre nous décrivons d'abord les objectifs de la validation expérimentale de notre solution. Ensuite, nous présentons les outils et méthodes expérimentales que nous avons mis en œuvre. En fin, nous présentons les résultats issus des expérimentations.

7.1 Objectifs

Les objectifs de la validation expérimentale peuvent être déclinés en deux points :

- **Valider la faisabilité de la solution**
  Nous validons la faisabilité de notre solution pour :
  -l'expressivité des requêtes d'interrogation des données de biodiversité.
  -le fonctionnement non-intrusif de notre architecture et la répartition dynamique des données à la demande.
- **Évaluer les performances**
  Nous évaluons les performances de notre solution pour :
  -la garantie des temps de réponse borné pour les requêtes.
  -le passage à l'échelle des sites.

7.2 Méthodes expérimentales

Les méthodes expérimentales de la validation concernent l'environnement, les outils utilisés, les expériences effectuées et les approches comparatives. Nous présentons dans les sous-sections suivantes les détails de nos méthodes expérimentales.

7.2.1 Environnement et outils

L'environnement expérimental correspond à l'ensemble des ressources matérielles et logicielles utilisées pour la réalisation de la validation expérimentale. En ce qui concerne les outils, ils désignent les données et les requêtes qui sont mises en œuvre dans les différentes expériences.

7.2.1.1 Environnement

Nous avons implémenté notre solution sur une infrastructure composée de onze machines. Chaque machine dispose des caractéristiques suivantes : 22 CPU de 2.394 GHz de fréquence, une mémoire de 62 Go avec comme système d'exploitation Debian 3.2.54-2 x 64. Une machine est utilisée pour représenter le portail GBIF et dispose de toutes les données dans sa base locale. Une deuxième machine est utilisée pour générer des requêtes et simuler des utilisateurs qui envoient des requêtes d'analyse au système décentralisé. Une machine représente le GAM (figure 3.4) pour contrôler les accès à la base de données du GBIF et gérer la cohérence des index. Les huit machines restantes sont utilisées pour instancier chacune un site du système décentralisé.

Chaque site dispose d'une base de données locale en plus des modules de gestionnaire local et gestionnaire de requêtes de notre prototype. Nous avons également implémenté les catalogues locaux et le catalogue global pour la localisation des données à travers le système réparti. Pour ce qui concerne les charges courantes des sites, elles sont demandées directement aux sites concernés. Comme SGBD, nous avons utilisé MonetDB [IGN+12]. MonetDB est un SGBD qui compresse et stocke les données en mémoire [IGN+12]. Il a pour vocation d'exploiter les grandes mémoires physique des systèmes de calcul modernes de façon efficace lors du traitement des requêtes. De ce fait, il offre de bonnes performances en évitant les accès disques souvent coûteux. En plus, il supporte plusieurs modèles de données pour la gestion des données. Par exemples il supporte le modèle de données relationnelles et SQL, le format XML et XQuery et le format RDF et SPARQL [IGN+12].

Pour ce qui concerne la gestion des communications entre les sites, nous avons utilisé le service de messagerie ActiveMQ [F.a] de Apache que nous avons installée sur une machine dédiée. Nous avons mesuré les durées des communications entre les machines via le service de messagerie. On constate elles sont inférieures à 5ms pour une communication entre deux machines.

7.2.1.2 Données

Nous avons utilisé dans nos expériences un miroir (dump) des données géo-référencées du GBIF que nous avons téléchargé et inséré dans la base MonetDB de la machine qui représente le portail

GBIF. A partir des données d'occurrences de la base, nous avons construit le catalogue taxonomique pour reprendre la structure hiérarchique des données de biodiversité sur la dimension taxinomique. Pour ce qui concerne l'organisation hiérarchique selon la dimension géo-spatiale, nous avons divisé le globe en neuf cellules représentant chacune un continent. Chaque continent est découpé en neuf cellules qui correspondent aux pays. Enfin, chaque pays est divisé en 9 régions (cellules). Chaque fragment est défini par une espèce et une région. Nous avons considéré les fragments ayant plus de 10000 occurrences dans la même région pour avoir des résultats plus significatifs.

7.2.1.3 Requêtes

Nous utilisons comme modèle de requêtes, le calcul de la cooccurrence de deux espèces dans une région pour toutes les expériences. Pour cela, une requête interroge deux fragments (définis par les prédicats taxinomique (nom de l'espèce) et géographique (numéro de la région)) de la même région. Nous avons implémenté un générateur de requêtes SQL qui produit au début de chaque expérience un nombre fixe de requêtes qui sont par la suite soumises au système.

7.2.2 Expériences

Les expériences menées dans le cadre de cette validation consistent dans un premier temps à étudier la faisabilité de notre solution décentralisée pour l'interrogation des données de biodiversité avec le mécanisme de répartition dynamique des données à la demande. Dans un deuxième temps, nous évaluons les performances de notre approche d'optimisation que nous avons comparée avec d'autres approches possibles présentées à la section 6.5. Nous étudions la garantie du passage à l'échelle de notre approche en analysant le comportement des différents sites pour les traitements des requêtes et l'adaptation de notre approche d'optimisation par rapport au contexte dynamique.

Pour chaque requête nous mesurons le temps de réponse, le temps de chaque traitement OL, le temps du traitement OG et le temps de chaque éventuelle réplication. Nous notons également tous les sites impliqués et leur tâche dans le traitement de la requête.

7.2.3 Approches comparatives

Les approches utilisées pour évaluer les performances de notre solution sont celles qui sont présentées à la section 6.5. Nous rappelons brièvement le principe d'optimisation de chacune d'elles.

- *MiniTrans :* correspond à l'approche qui minimise le coût de transfert pour une requête en répliquant les données manquantes vers le site qui dispose de la plus grande portion de données impliquées par la requête. Pour cela, cette approche classe les sites par quantité de données de la requête hébergées. Le site qui a la plus grande quantité de données est choisi pour coordonner et traiter la requête. Il réplique les données de la requête dont il ne dispose pas vers sa base locale à partir d'un voisin où du GBIF lorsque la donnée est manquante des sites locaux. Lorsque les données de la requête ne sont sur aucun site, alors le site qui l'a reçu est choisi pour coordonner et traiter la requête.
- *EquiLoad :* a pour objectif de minimiser les attentes pour les requêtes. Pour cela, elle choisit le site le moins chargé pour coordonner le traitement de la requête en répliquant localement les données dont il ne dispose pas. Puis, ce coordinateur traite localement la requête. Contrairement à MiniTrans même si toutes les données de la requête sont disponibles uniquement au GBIF, le coordinateur reste le site le moins chargé du système.
- *DisNoRep :* a pour objectif d'exploiter les avantages du traitement réparti et réduire par conséquent les transferts de données entre les sites locaux. Pour cela, cette approche traite chaque fragment sur le site local qui dispose de l'unique réplique. Il importe de noter que DisNoRep ne réplique pas de fragment à partir d'un site local.

7.3 Faisabilité

Pour valider la faisabilité de notre solution, nous l'étudions d'une part par rapport aux problèmes qui ont motivé notre travail. Rappelons que les problèmes identifiés dans le fonctionnement actuel du GBIF peuvent être résumés en deux points :

i) *L'expressivité limitée* des requêtes supportées par le portail réduit les possibilités d'exploitation des données de biodiversité. En d'autres termes, les fonctionnalités d'analyse des données de biodiversité sont réduites.

ii) *L'architecture centralisée* pour l'interrogation est une source de congestion. De ce fait, elle ne garantit pas une disponibilité permanente du système lorsque le nombre et la fréquence des accès des utilisateurs accroissent. Cela peut poser de réels problèmes de performance pour la réactivité du système vis-à-vis de ses usagers surtout pour le passage à l'échelle des fonctionnalités (données et traitements).

7.3.1 Expressivité des requêtes

Notre solution permet de résoudre le problème de l'expressivité des requêtes. En effet, on peut traiter des requêtes écrites en SQL avec le support des fonctions d'agrégation. En outre, notre modèle de requêtes permet d'exprimer les opérations nécessaires pour des requêtes d'analyse de

biodiversité : sélection, agrégation, jointure, création de tables temporaires etc. Ce modèle de requêtes de calcul de cooccurrence qui est mis en œuvre dans nos expériences, a été évalué pour différentes requêtes et différents prédicats portant sur la taxonomie et la géographie avec des temps de réponse raisonnables. La seule contrainte qui se pose dans notre modèle de requêtes est la requête doit avoir des prédicats sur la taxinomie et la géographie. Puisque les requêtes d'analyse consistent généralement à étudier un taxon dans une zone géographique bien définie, alors on suppose que chaque requête dispose de prédicats sur la taxonomie et sur la géographie.

Les performances de notre solution pour le traitement de requêtes qui sont détaillées dans la section 7.4, montrent que notre solution supporte des requêtes plus expressives.

7.3.2 Architecture décentralisée non-intrusive : Répartition dynamique des données à la demande

Pour résoudre les problèmes liés à l'accès centralisé aux ressources de biodiversité partagées à travers le réseau du GBIF, nous avons proposé un système réparti décentralisé dans lequel les données sont réparties dynamiquement à la demande en fonction des prédicats des requêtes et de la structure hiérarchique des données selon les dimensions taxinomique et géographique. Nous avons implémenté le système décentralisé proposé et le mécanisme de répartition dynamique des données à la demande à travers les nœuds de ce système. Rappelons que dans le fonctionnement de notre solution, on accède au GBIF seulement lorsqu'une donnée nécessaire au traitement de la requête n'est pas disponible dans un site local et qu'au démarrage du système, toutes les bases locales sont vides.

L'objectif de notre solution décentralisée est de décharger le GBIF des tâches d'interrogation des données et de réduire progressivement les accès à la base du portail. Pour ce qui concerne le traitement de requêtes, notre solution traite toutes les requêtes à travers les sites locaux. Et la base du GBIF est utilisée uniquement pour lire des données manquantes (qui n'existent pas encore dans un site local) et nécessaires à l'exécution d'une requête en cours.

7.3.2.1 Réduction progressive des temps d'accès au portail GBIF

Nous évaluons la progression des temps d'accès au GBIF par intervalle de temps au cours de l'expérience. Pour cela, nous récapitulons pour chaque requête, la durée totale des transferts (ou réplication) de fragments issus du GBIF vers les sites locaux pendant chaque intervalle de temps de dix minutes au cours de l'expérience.

Les résultats représentés sur le graphe 7.1, nous permettent de constater un temps élevé pour les accès à la base au GBIF dans les premiers instants de l'expérience. Ce qui est tout à fait normal, d'autant plus qu'au démarrage de l'expérience aucune donnée n'est disponible dans une base

locale. D'où la nécessité d'aller chercher à la demande les données manquantes invoquées par les requêtes au GBIF et de les répliquer dans les bases locales. Ce placement des données à la demande a permis d'utiliser ces données pour les prochaines requêtes. Par la suite, on constate que les temps d'accès à la base du GBIF diminuent progressivement jusqu'à s'annuler vers la fin de l'expérience.

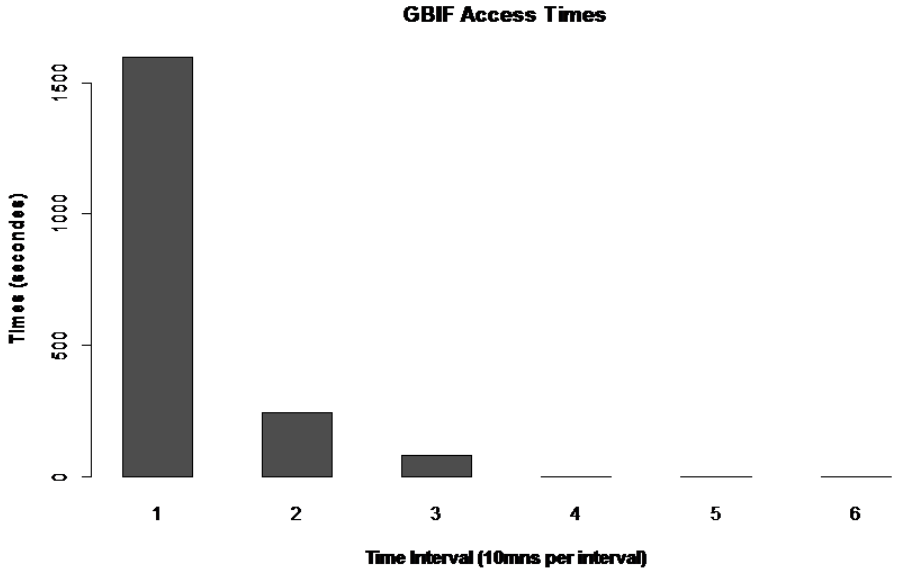

FIGURE 7.1 – Evolution des temps d'accès au portail GBIF par période

Cette réduction progressive des temps d'accès au GBIF pour les requêtes, montre l'efficacité de notre approche de répartition dynamique à la demande des données en fonction des requêtes. En effet, notre mécanisme de répartition des données à la demande permet de répartir la durée de migration des données dans la période d'exploitation du système. Ce qui favorise le traitement de requêtes dans le système décentralisé en parallèle avec la migration de nouvelles données venant du GBIF à travers les sites qui composent ce système. En outre, cette politique de répartition à la demande permet de réduire la quantité de données à placer sur les sites locaux lorsque les requêtes ne concernent pas toutes les données de la base du GBIF. Ceci réduit non-seulement la durée globale de la migration mais sauvegarde aussi de l'espace de stockage à travers le système décentralisé. La capacité de stockage cumulée des sites locaux dépasse la taille de la base du GBIF. L'expérience s'est déroulée avec des requêtes accédant à 300 fragments qui représentent 20% de la base. Si par la suite des nouvelles requêtes interrogent d'autres fragments, d'autres accès au GBIF seraient nécessaires. Cependant, les accès se réduisent globalement dès que les fragments demandés sont disponibles dans les sites locaux.

7.3.2.2 Adaptation dynamique du schéma de réplication des données

Nous évaluons la variation du degré de réplication des fragments en fonction de leur fréquence d'accès pendant une expérience. Pour cela, nous incrémentons pour chaque fragment son nombre d'accès à chaque fois qu'il est impliqué dans une requête. Nous connaissons aussi le degré de réplication de chaque fragment. Après avoir trié les fragments par leurs nombres d'accès, nous représentons le nombre d'accès (figure 7.2) et le degré de réplication (figure 7.3) de chaque fragment, puis nous les regroupons par classe de 15 fragments en fonction des nombres d'accès et représentons le degré de réplication moyen (figure 7.4)de chaque classe.

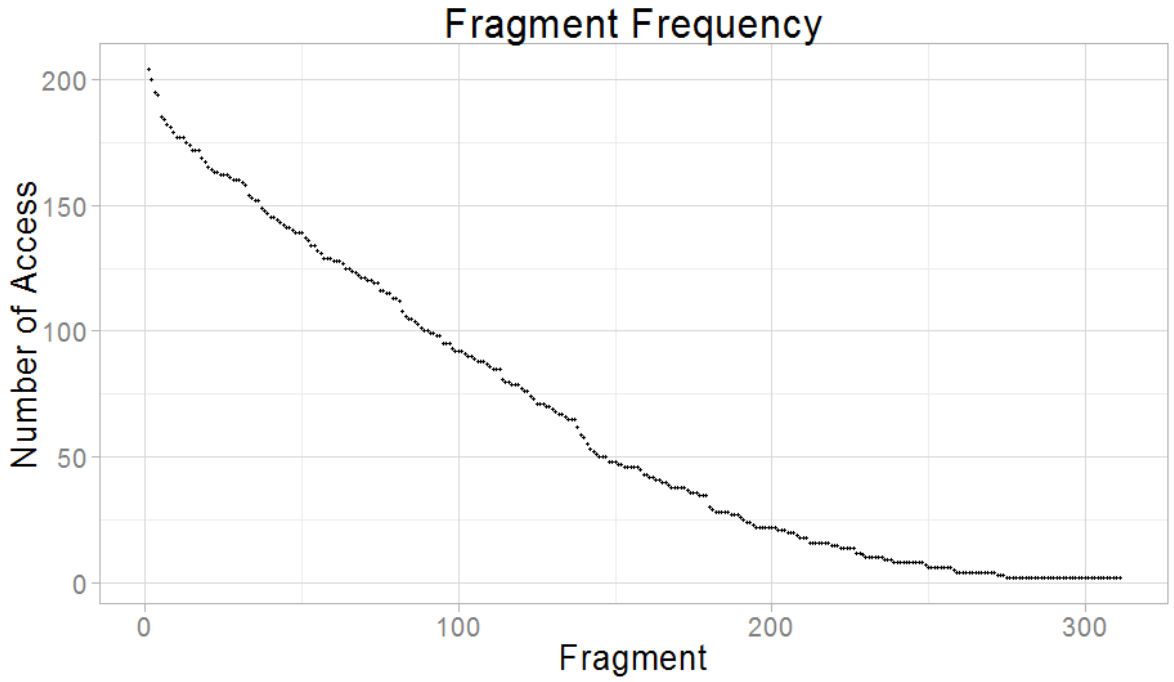

FIGURE 7.2 – Popularité des fragements

Les figures 7.2, 7.3 et 7.4 nous permettent de réduire que le degré de réplication de chaque fragment (ou classe de fragments) évolue généralement dans le même sens de la popularité du fragment (ou de la classe). Ce qui prouve que notre mécanisme de répartition adapte le degré de réplication de chaque fragment en fonction de la quantité de requêtes qui l'invoquent. Cette adaptation du degré de réplication de chaque fragment en fonction de sa popularité permet d'éviter des coûts de stockage inutiles tout en garantissant la disponibilité des données. Par ailleurs, on voit que certains fragments de la classe 5 sont légèrement moins répliqués que des fragments moins sollicités de la classe 6 sur la figure 7.4. Ceci s'explique par le fait que la décision de répliquer un fragment dépend aussi de la surcharge des sites qui hébergent les répliques du fragment. De ce fait, un fragment très sollicité avec des répliques qui se trouvent sur des sites non-surchargés aux moments où les requêtes qui l'invoquent sont analysées, n'est pas répliqué.

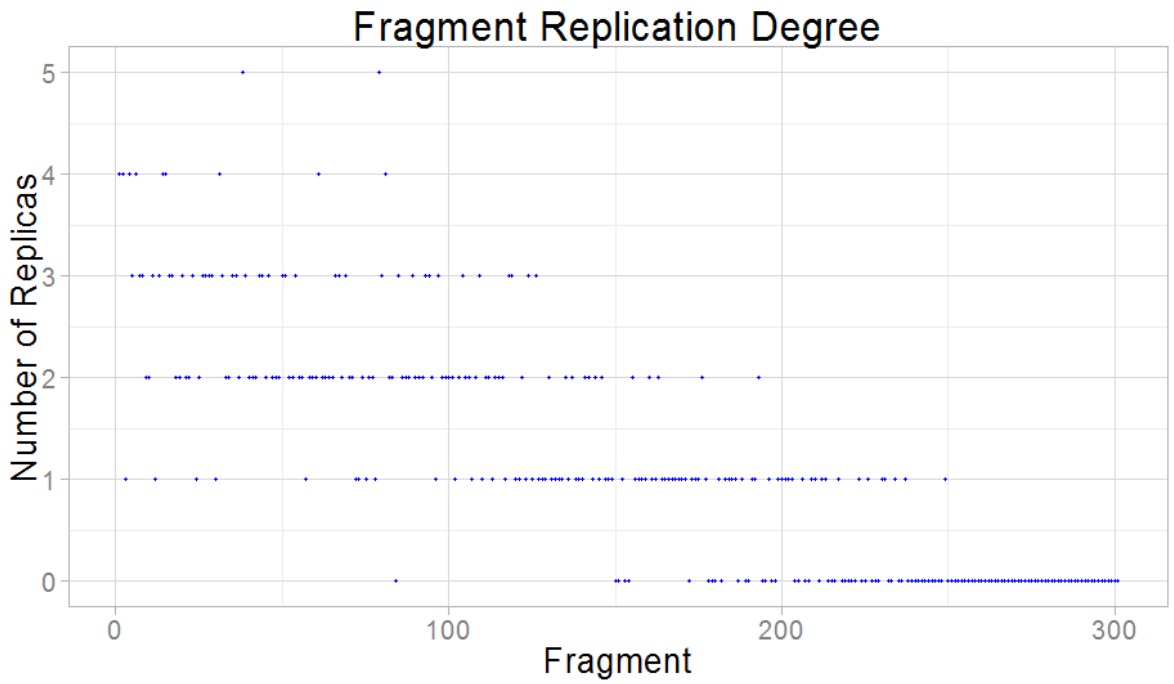

FIGURE 7.3 – Dégrés de réplication des fragements

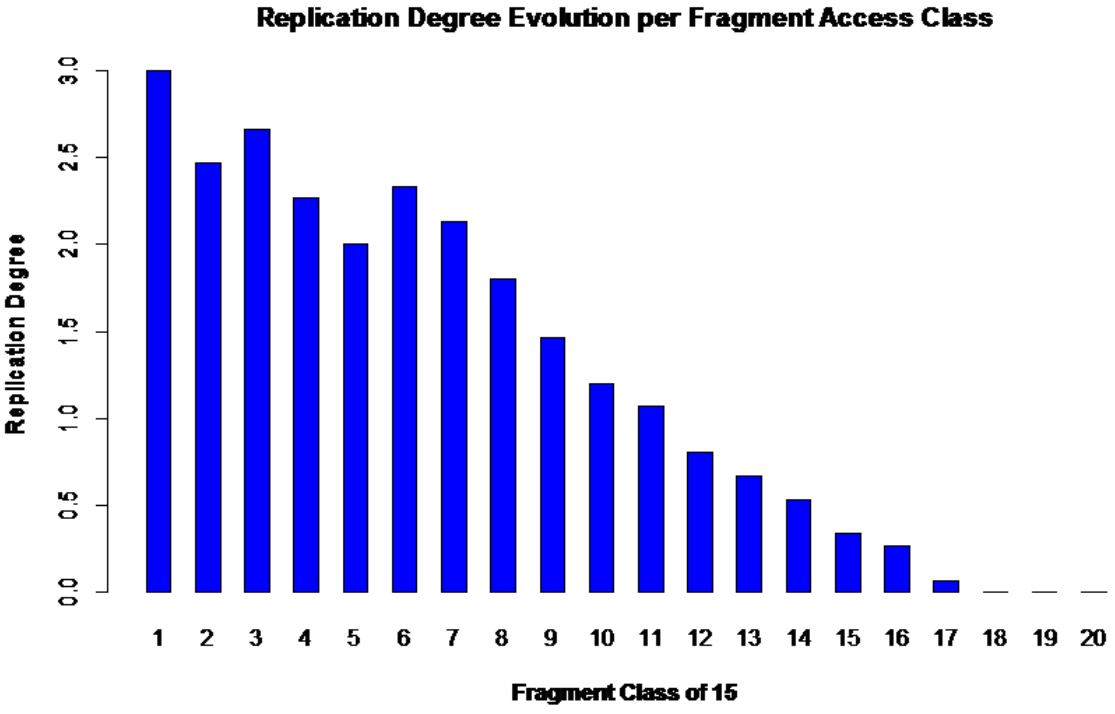

FIGURE 7.4 – Dégré de réplication par classe d'accès

7.4 Evaluation des performances

L'objectif de cette section consiste à évaluer le débit (i.e. le nombre de requêtes traitées par minute), que fournit notre solution tout en garantissant un temps de réponse borné.

Nous évaluons les performances en fonction de l'évolution du nombre de requêtes traités par intervalle de temps et de l'approche d'optimisation mise en œuvre. Par la suite, nous comparons les performances de notre solution appelée optipara avec les approches Minitrans, Equiload et Disnorep dont les fonctionnement sont expliqués à la section 7.2.3.

Nous avons divisé la durée de chaque expérience en intervalles de temps de dix minutes et avons récapitulé le nombre de requêtes traitées dans chaque intervalle pour chaque approche. Les différentes expériences sont effectuées dans les mêmes conditions : même nombre de machines, même nombre d'utilisateurs et même nombre de requêtes. Nous représentons sur les figures uniquement les périodes complètes communes à toutes les expériences. La figure 7.5 représente les évolutions des débits des différentes approches en fonction du temps.

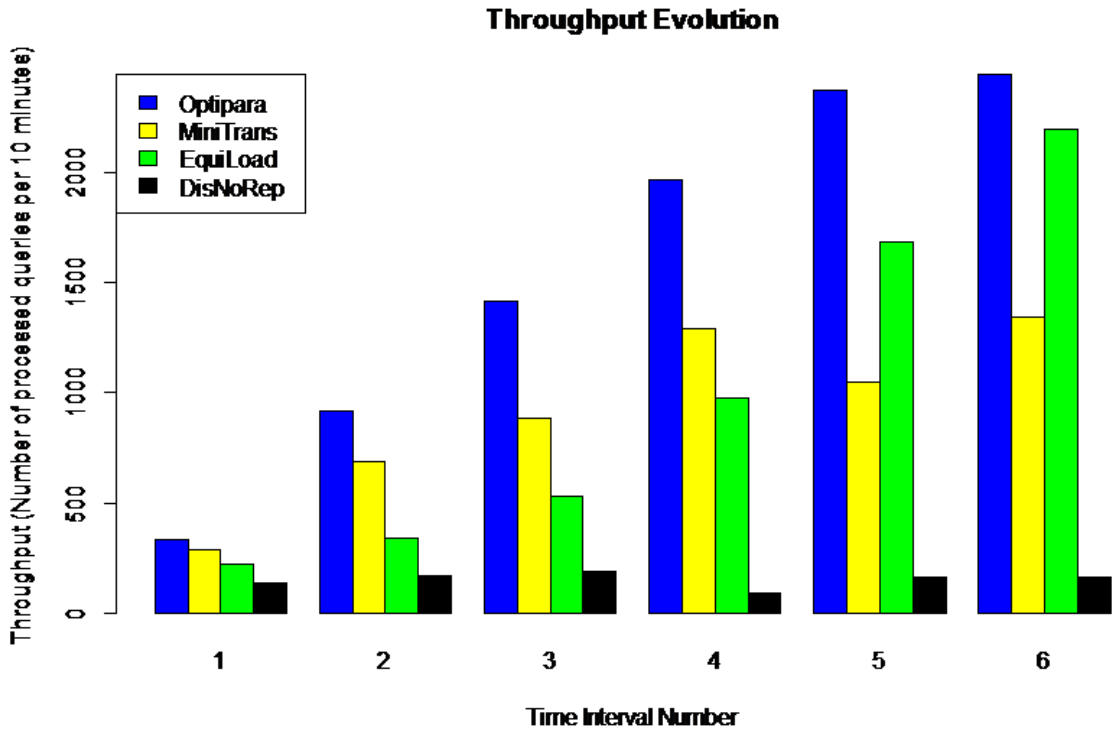

FIGURE 7.5 – Evolution des performances des différentes approches

Nous constatons que l'approche DisNoRep possède les plus faibles débits dans toutes les périodes. Ceci s'explique par le fait qu'elle est la seule approche qui ne réplique pas de fragment. De ce fait, même si elle favorise le traitement parallèle, les accès concurrents aux mêmes fragments

surchargent les sites qui les contiennent et par conséquence dégradent les performances. Puisque le schéma de placement de chaque fragment est statique, les performances ne sont pas améliorées au cours du temps.

Contrairement à DisNoRep, toutes les autres approches voient leurs débits augmenter au cours du temps. Ces approches adoptent un mécanisme de réplication dynamique de fragments invoqués dans les requêtes. Ce qui donne davantage de possibilités à l'optimiseur de requêtes pour le calcul du plan d'exécution. Cependant, les approches MiniTrans et EquiLoad n'exploitent pas le parallélisme intra-requête qu'aurait offert le traitement réparti pour éviter certains transferts inutiles. L'augmentation de leurs débits s'explique par la prise en compte de paramètres importants qui ont un impact considérable sur le coût de la requête. En effet, MiniTrans réduit les transferts en répliquant les données manquantes au site qui dispose de la plus grande portion de données impliquées. Ce qui fait les données d'une requête sont regroupées en un seul site et

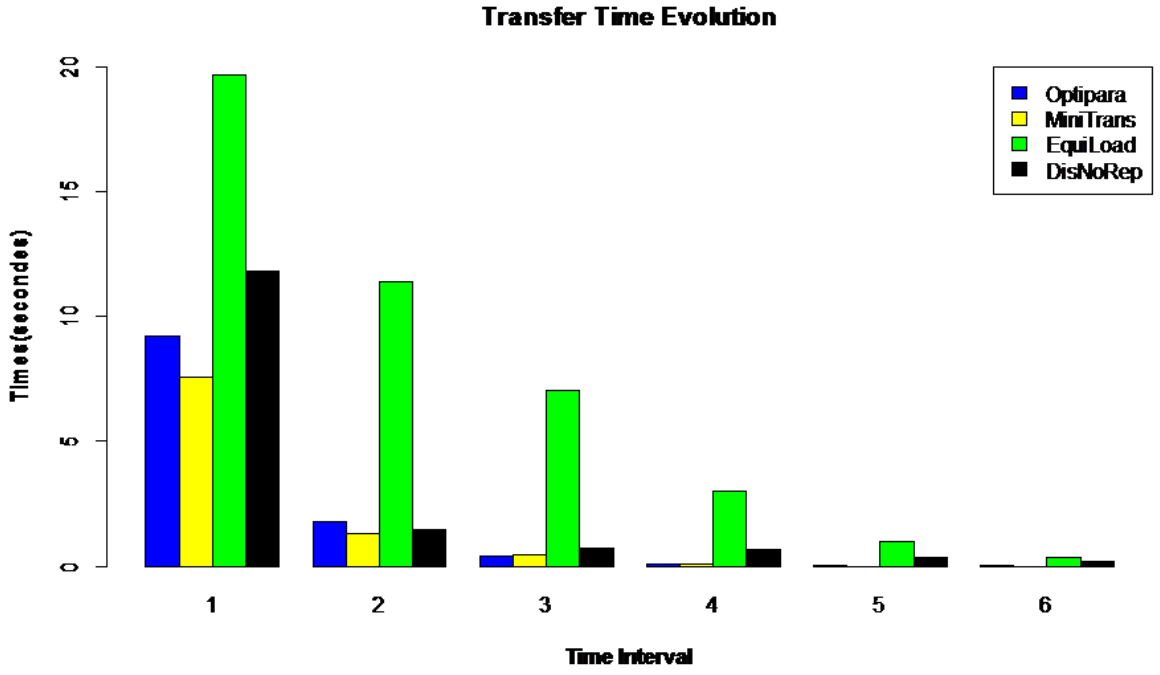

FIGURE 7.6 – Evolution des temps de réplication

que les prochaines qui invoquent les mêmes données ne nécessiteront pas de transfert. Ainsi les transferts de données pour le traitement de requêtes sont de moins en moins fréquents dans le système. C'est ce qui explique l'augmentation progressive du débit. Cependant cet accroissement est limité par le fait que MiniTrans utilise un petit nombre de sites regroupant la totalité des données invoquées par les requêtes. Par conséquent seuls ces sites sont utilisés pour les traitements des requêtes. De ce fait, MiniTrans n'équilibre pas la charge entre les sites lorsque les sites qui disposent des données sont surchargés.

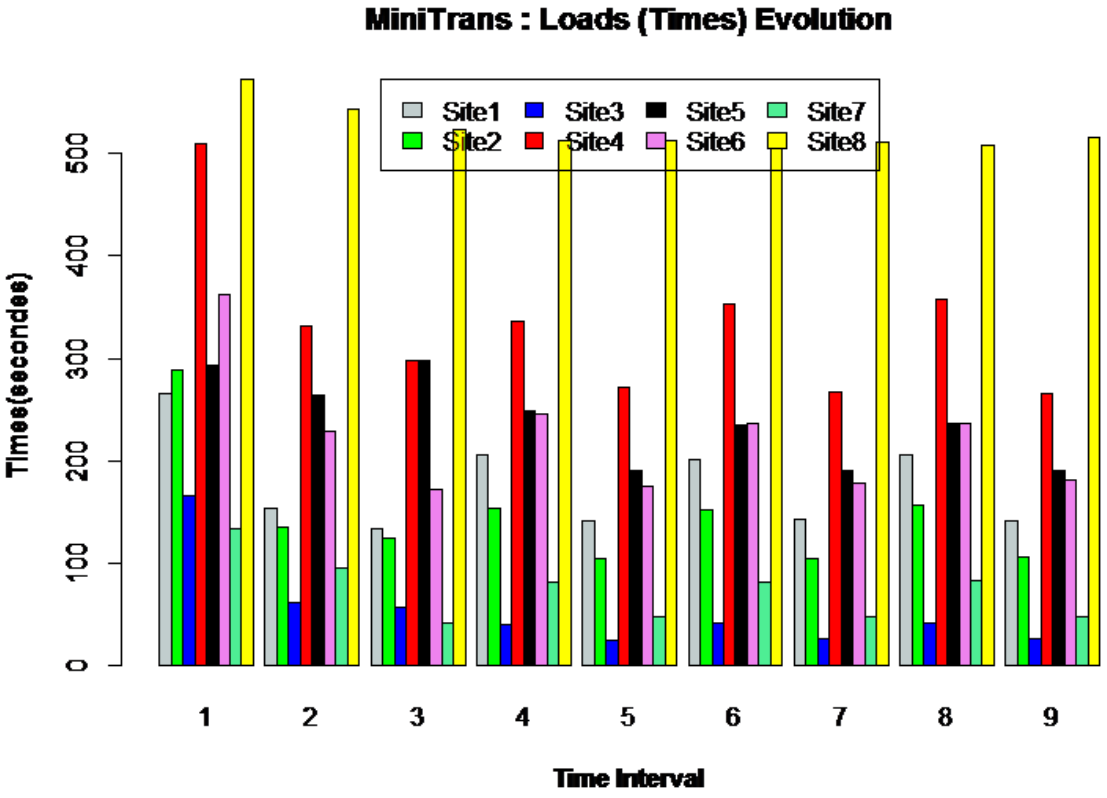

FIGURE 7.7 – Evolution de la charge à travers les sites pour MiniTrans

En ce qui concerne l'approche EquiLoad, elle règle ce problème d'équilibrage de la charge en traitant la requête sur le site le moins chargé et permet ainsi de réduire le temps d'attente pour une requête et de répartir de façon quasi-équitable la charge globale à travers les sites (figure 7.8. L'accroissement progressif de son débit s'explique par la réplication des données manquantes vers le site choisi pour traiter la requête. En effet, cette réplication dynamique crée davantage des répliques pour les fragments très sollicités et augmente ainsi les possibilités de choix pour les prochaines requêtes qui les sollicitent. Ce qui permet de réduire davantage les transferts pour les requêtes invoquant des fragments fréquents. Cependant, elle ne peut aboutir à une solution de réplication totale qui élimine complètement les transferts de données puisque la capacité de stockage de chaque base locale est limitée. Ce qui fait que les transferts de données restent toujours présents.

Notre approche, OptiPara, combine les avantages des trois approches en considérant la charge des sites, le coût de transfert et exploitant les possibilités du traitement réparti pour une requête. En effet, notre approche s'appuie sur une contrainte de temps de réponse borné que le plan d'exécution de la requête doit respecter si cela est possible. Lorsque cette contrainte ne peut être respectée, notre approche en profite pour adapter dynamiquement le schéma de placement

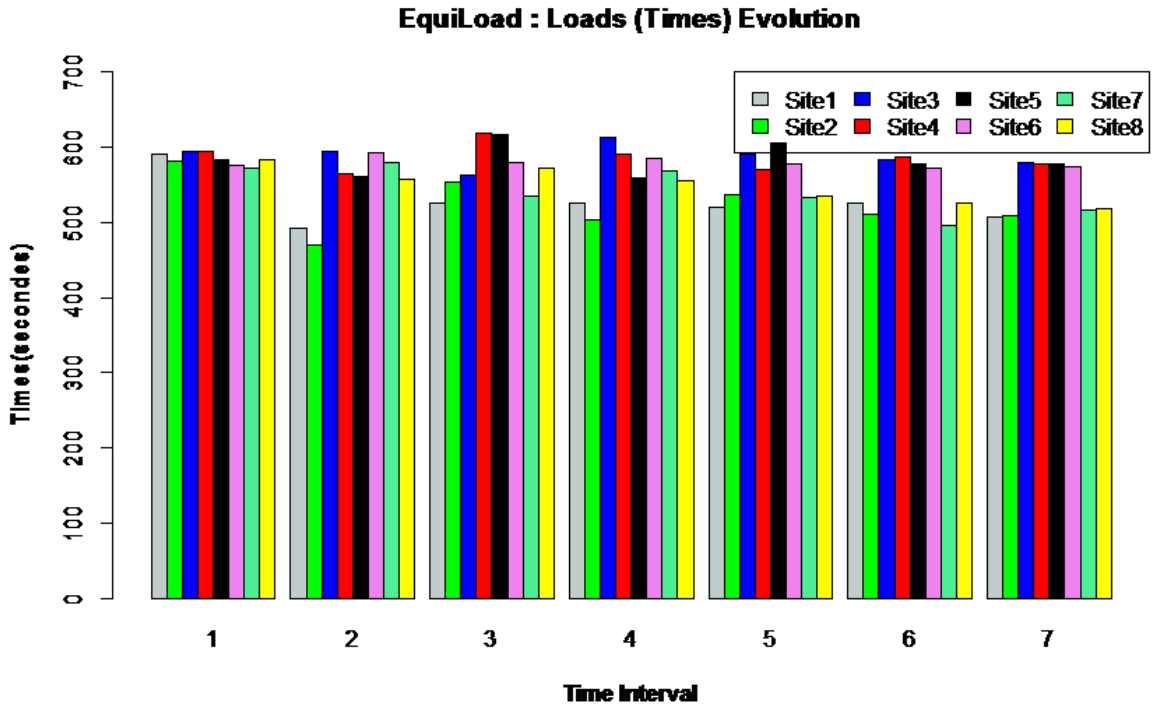

FIGURE 7.8 – Evolution de la charge à travers les sites pour EquiLoad

en répliquant les données qui se trouvent sur des sites surchargés (qui ne peuvent pas garantir un temps de réponse borné pour une nouvelle requête) vers des sites moins chargés. De cette façon, elle évite de surcharger les sites. La répartition de la charge entre les sites dépend donc de la capacité de chaque site. Cette stratégie ne garantit pas une charge équitable entre les sites, mais elle évite qu'un site soit surchargé en attribuant les nouveaux traitements aux sites libres qui peuvent garantir un temps de réponse borné.

La réplication dynamique des données offre plus de possibilités pour les calculs des plans d'exécution des requêtes qui invoquent les mêmes données. Puisque notre approche supporte la possibilité du traitement réparti d'une requête, alors les transferts (réplication) de fragments qui se trouvent sur des sites non-surchargés (pouvant respecter la contrainte de temps de réponse borné) ne sont pas nécessaires et par conséquent sont évités. Notre modèle d'optimisation basé sur une contrainte de temps de réponse borné est efficace dans la mesure où il augmente progressivement le pourcentage (figure 7.11) des requêtes qui respectent (figure 7.10) cette contrainte.

De plus, notre approche d'optimisation de requêtes favorise progressivement la localité du traitement des requêtes en minimisant les participants. La figure 7.12 montre l'évolution de la localité par intervalle de temps des requêtes en fonction de l'évolution du temps pendant l'expérience.

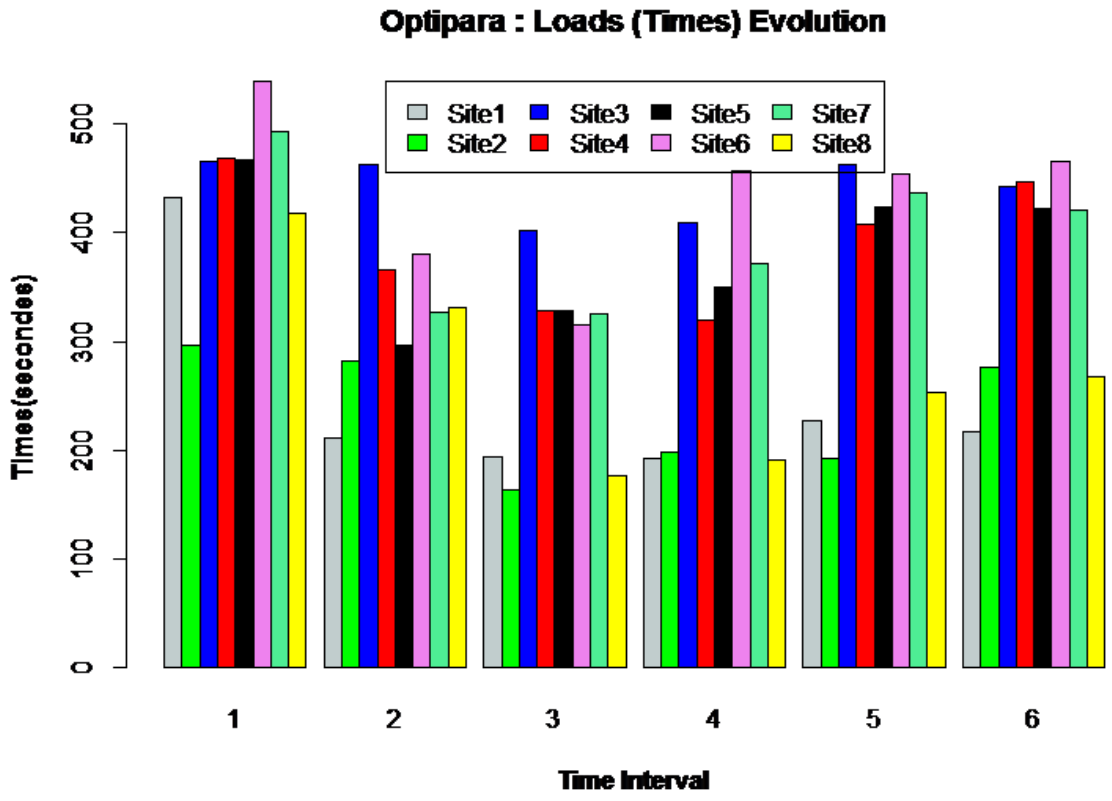

FIGURE 7.9 – Evolution de la charge à travers les sites pour notre approche

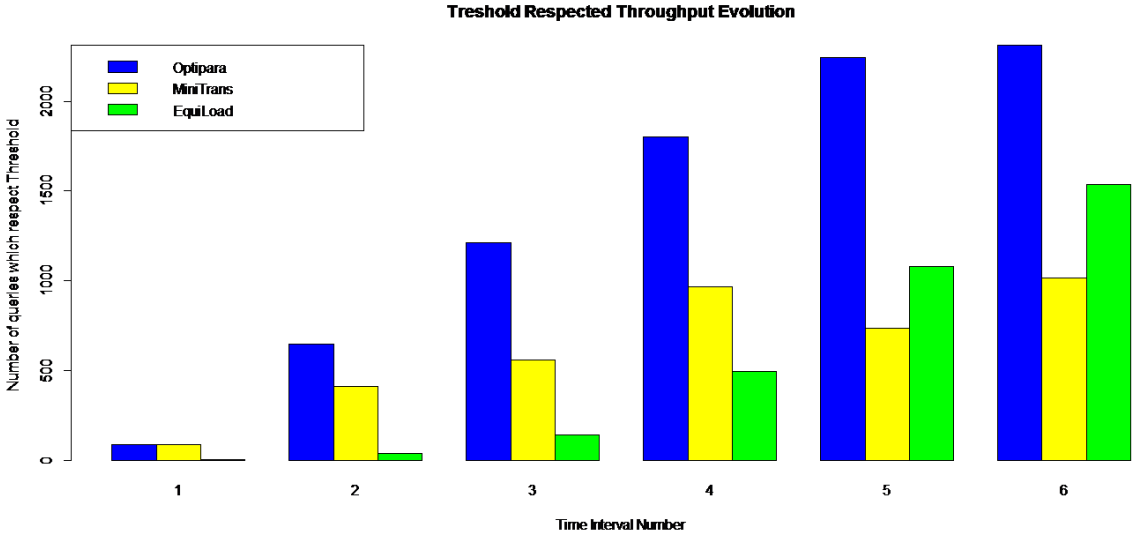

FIGURE 7.10 – Evolution du débit des requêtes qui respectent le seuil

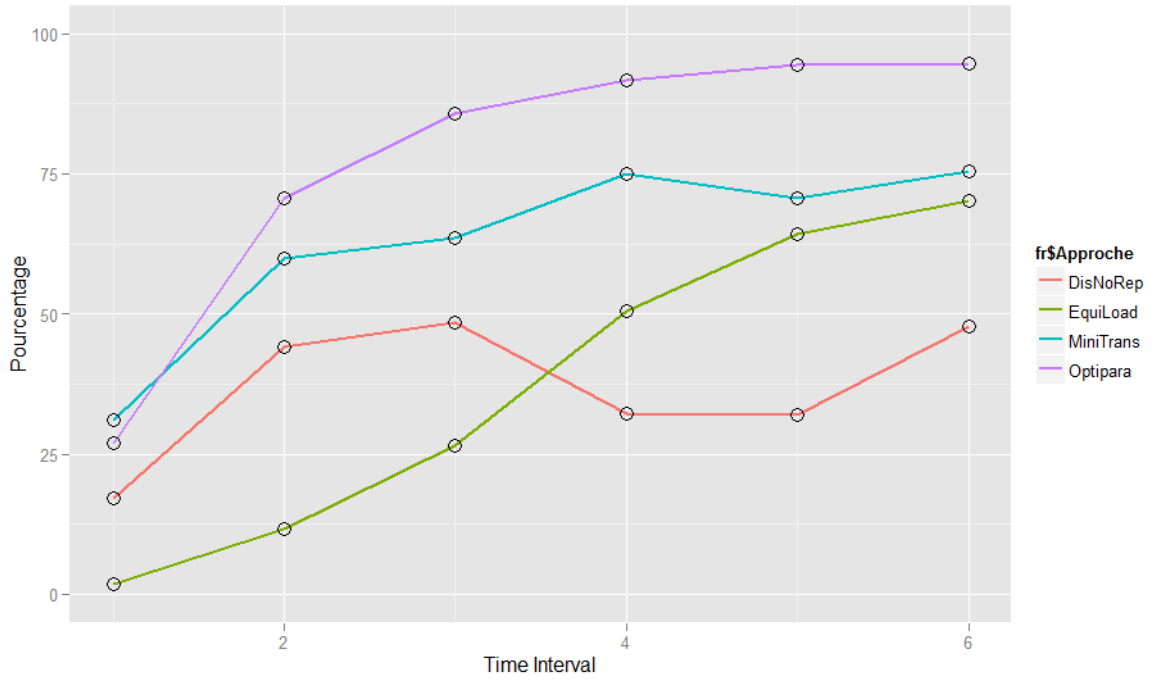

FIGURE 7.11 – Evolution du pourcentage des requêtes qui respectent le seuil

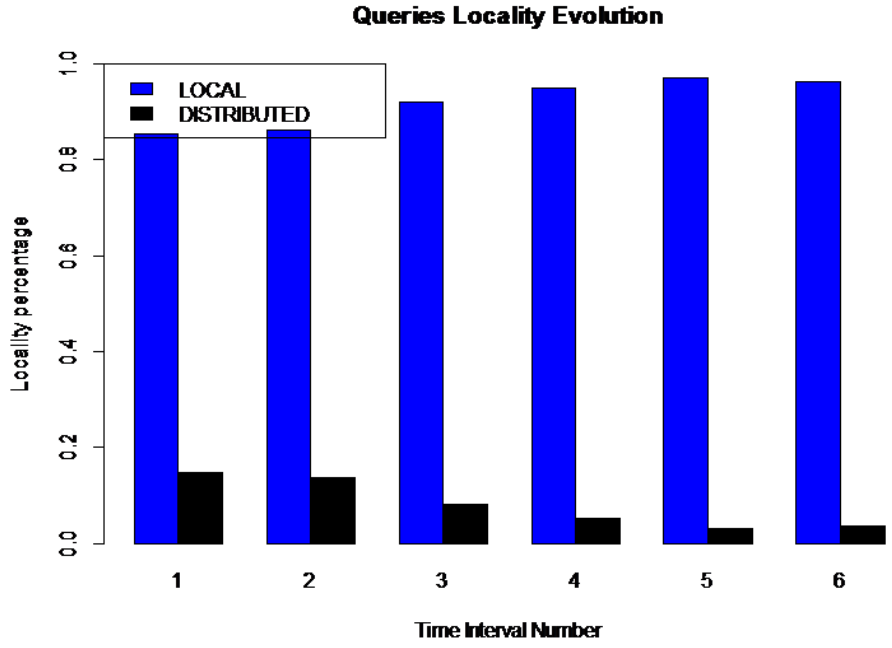

FIGURE 7.12 – Evolution de la localité des requêtes

7.5 Passage à l'échelle

L'objectif de cette validation du passage à l'échelle est d'étudier les performances de notre solution en fonction de l'évolution des ressources et de la charge. Idéalement, une solution garantit un passage à l'échelle si les performances doublent quand les ressources doublent.

Nous évaluons la garantie du passage à l'échelle de notre solution en évaluant les performances dans des scénarios différents. Chaque scénario est caractérisé par le nombre de sites utilisés et la charge soumise au système. Le nombre de sites et la charge d'un scénario à un autre varient avec le même facteur de proportionnalité. Pour chaque scénario nous notons ses débits stable et global. Le débit stable correspond à la moyenne des débits par intervalle de temps à partir desquels les variations sont inférieures à 5%.

Nous récapitulons dans le graphe 7.13 les évolutions des débits par minute en fonction du nombre de sites qui composent le système décentralisé.

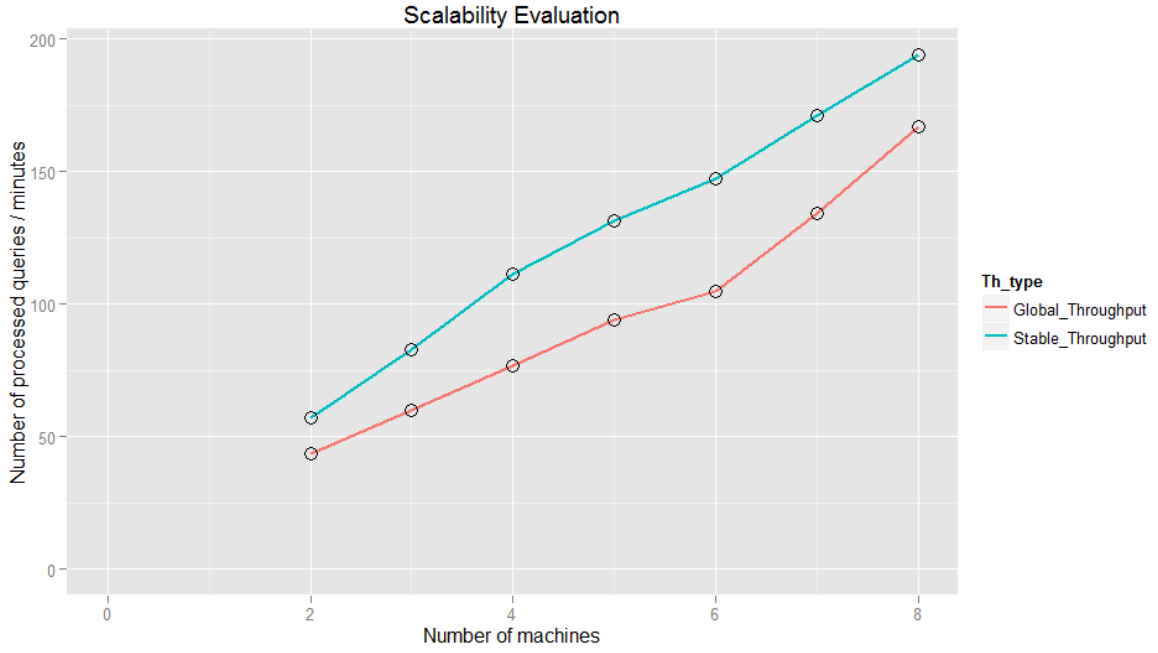

FIGURE 7.13 – Evolution des performances en fonction du nombre de sites

Nous constatons que le débit augmente proportionnellement avec le nombre de sites. En effet, le débit évolue dans le même sens que le nombre de sites qui composent le système et avec la même valeur. En outre, on voit que le débit stable est meilleur que le débit global. Ceci s'explique par le fait que le système nécessite une phase d'adaptation qui peut engendrer des réplications de données. L'adaptation dynamique du schéma de placement des données permet de réduire progressivement les transferts (ou réplications) pour les prochaines requêtes. De ce fait, avant

la stabilisation du système, les sites, en plus des traitements, coordonnent aussi les réplications de données. Ce qui influe sur les performances globales du système. Lorsque le système est stable, alors les transferts dûs aux réplications deviennent de plus en plus rares et les sites se concentrent entièrement aux traitements des requêtes. C'est ce qui explique que les débits stables sont supérieurs aux débits globaux. Avec cette adaptation dynamique du schéma de placement des données et le mécanisme d'optimisation basée sur le coût, notre solution garantit le passage à l'échelle.

7.6 Résultats des tests avec Spark

Nous avons effectué quelques expériences avec Spark pour évaluer le temps de réponse d'une requête. Pour cela, nous avons utilisé le même modèle de requêtes que nos expériences et le même cluster de machines. Nous avons utilisé Spark pour répartir les données avant de poser les requêtes avec le langage scala. Le temps de réponse moyen obtenu avec Spark est 11 fois plus élevé que celui de notre solution.

7.7 Conclusion

Dans ce chapitre, nous avons présenté la validation expérimentale de notre solution décentralisée. Nous avons instancié une architecture répartie avec tous ses modules. Nous avons préparé les données pour observer des résultats significatifs. Nous avons mis en œuvre le modèle de requêtes de cooccurrence de deux espèces dans une même région géographique. Ceci nous a permis de valider la faisabilité de notre solution pour résoudre les problèmes d'expressivité des requêtes d'analyse de biodiversité.

Nous avons validé la faisabilité de la répartition dynamique des données à la demande en fonction des prédicats des requêtes. Pour cela, nous avons étudié l'évolution des temps d'accès au portail pour lire des données et l'adaptation dynamique du degré de réplication de chaque fragment. L'évolution des temps d'accès au portail montrent que les données sollicitées par les requêtes sont davantage disponibles dans les bases locales et que notre solution accède de moins en moins au portail pour traiter les requêtes des utilisateurs. Ceci permet de décharger du portail les tâches d'interrogation des données de biodiversité. Les résultats montrent aussi que notre solution adapte dynamiquement le placement des données en fonction des requêtes et des charges des sites.

Nous avons évalué les performances de notre approche d'optimisation basée sur le coût en tenant compte des paramètres de charges, de transfert, des ressources (données et calcul) des sites.

Les résultats obtenus sont satisfaisants et sont meilleurs que les performances des approches d'optimisation plus simples telles que MiniTrans, EquiLoad et DisNoRep.

Nous avons également étudié la garantie du passage à l'échelle pour notre solution en évaluant les performances pour des systèmes qui diffèrent par le nombre de sites qui les constituent. Les performances obtenues évoluent dans le même sens que le nombre de sites avec le même rapport. Ceci prouve la garantie du passage à l'échelle pour notre solution.

Chapitre 8

Conclusion et Perspectives

8.1 Conclusion

Cette thèse aborde l'interrogation de données de biodiversité dans un contexte fortement contraint imposant d'utiliser des machines de petites capacités et dispersées à travers le monde. Nous avons tout identifié le cas d'usage principal caractérisant l'interrogation des données de biodiversité. Nous avons proposé une architecture décentralisée pour l'interrogation de données de biodiversité permettant de garantir la disponibilité du système en présence d'une quantité croissante de données et de traitements. Après avoir présenté les principes généraux de notre solution et les composants de son architecture répartie, nous avons proposé une stratégie de répartition dynamique des données de biodiversité. Ensuite, nous avons décrit le mécanisme de traitement des requêtes identifiées dans le cas d'usage principal. Puis, nous avons proposé une solution d'optimisation de requêtes basée sur un modèle de coût dynamique. Enfin, nous avons validé expérimentalement notre solution avec des données réelles de biodiversité issues du GBIF et des requêtes d'analyse d'occurrences.

Dans cette thèse nous avons développé les points suivants :

Architecture décentralisée non-intrusive

Notre architecture s'interpose entre les usagers et le portail du GBIF. Elle est chargée de traiter les requêtes des utilisateurs en répartissant à la demande les données du GBIF dans les sites locaux. De ce fait, tous les traitements d'interrogation sont décentralisés à travers cette architecture répartie. Il importe de noter que la solution est non-intrusive dans la mesure où elle ne modifie pas le fonctionnement actuel du portail.

Les sites qui constituent le système décentralisé mutualisent leurs ressources de calcul et de stockage pour garantir la disponibilité du service de requêtes. En effet, chaque site dispose d'un

SGBD qui lui permet de traiter des requêtes venant des utilisateurs ou des sites voisins. Un site local peut aussi traiter une sous requête venant d'un voisin ou répliquer localement des données nécessaires aux traitements qui lui sont assignés.

Répartition dynamique des données de biodiversité

Notre approche de répartition dynamique des données de biodiversité à la demande à travers l'architecture décentralisée, exploite la structure hiérarchique des dimensions taxinomique et géographique des données de biodiversité et les spécificités des requêtes d'analyse pour fragmenter les données. Ensuite, nous avons proposé un mécanisme de réplication des données, qui est guidé par le processus d'optimisation de la requête. Afin de garantir la disponibilité des données sollicitées par les requêtes qui se trouvent sur des sites avec des capacités de stockage limitées, nous avons proposé une stratégie de remplacement de données qui s'adapte au contexte de la répartition dynamique des données. Nous avons aussi proposé un mécanisme d'indexation hybride qui combine un index global et des index locaux pour localiser rapidement les données impliquées dans le traitement d'une requête. Pour garantir la cohérence des données disponibles dans le système décentralisé, nous avons proposé un mécanisme de propagation asynchrone des mises à jour effectuées sur la base du GBIF vers les sites locaux.

Traitement réparti de requêtes d'analyse

Nous avons proposé un modèle de requêtes d'analyse de données de biodiversité qui généralise le calcul de la cooccurrence de plusieurs taxons dans une même zone géographique. Ensuite, nous avons proposé une solution de traitement réparti de requêtes d'analyse en poussant au plus possible les opérations locales aux fragments de la requête vers les données qui peuvent être réparties. Nous avons détaillé les différentes étapes du traitement d'une requête, de la réception au retour du résultat. Pour cela, nous avons présenté le processus d'analyse de la requête et les principes généraux de l'optimisation. Puis nous avons décrit les éventuelles réplications de données qui peuvent être nécessaires pour le traitement de la requête. Ces réplications s'effectuent du portail vers un site local ou entre deux sites locaux.

Optimisation de requêtes basée sur un modèle de coût dynamique

Nous avons proposé une approche d'optimisation de requêtes dans un environnement réparti à large échelle. Ce type d'environnement est caractérisé par les charges dynamiques, la disparité des performances des sites, des capacités des liens de communication inter-sites et des tailles des fragments interrogés dans les requêtes, et des fréquences d'accès aux fragments disparates et fluctuantes. Notre approche d'optimisation qui a pour objectif de garantir la disponibilité du service de requêtes est basée sur un modèle de coût dynamique. Avec une contrainte de temps de réponse borné à respecter pour la requête, notre approche d'optimisation évite de surcharger un site. Pour cela, elle utilise les fragments existants lorsqu'ils peuvent garantir la contrainte de temps de réponse borné. Dans le cas contraire, elle propose de répliquer les fragments de la

requête qui se trouvent sur des sites surchargés vers des sites qui peuvent garantir leur traitement dans les délais. Le choix de la destination d'une nouvelle réplique est guidée par l'objectif de garantir un temps de réponse borné pour la requête et de regrouper davantage les données accédées ensemble. De ce fait, on voit que notre approche d'optimisation de requêtes adapte dynamiquement le placement des données en fonctions des requêtes et des charges courantes des sites. Le modèle de coût de notre approche d'optimisation intègre plusieurs paramètres qui influencent le temps de réponse d'une requête à savoir : la charge et les performances de chaque site impliqué, la localisation et le coût de traitement de chaque fragment impliqué sur le site qui le traite, les durées de transferts des données entre les sites et les propriétés des fragments et des opérations de la requête. Nous avons présenté les algorithmes qui calculent le plan d'exécution d'une requête.

Validation expérimentale

Nous avons effectué la validation expérimentale de notre solution décentralisée non-intrusive avec des données réelles issues de la base complète du GBIF sur un cluster composé d'une dizaine de machines (200 cœurs). Comme modèle de requêtes d'analyse de biodiversité, nous avons utilisé la requête de calcul cooccurrence de deux espèces dans une même région géographique. Les résultats de nos expériences prouvent la faisabilité de notre solution en permettant le traitement de requêtes complexes qui ne sont pas supportées actuellement sur le portail du GBIF. En plus, ils montrent que notre solution est non-intrusive car les requêtes sont traitées à travers les sites locaux et que notre stratégie de répartition dynamique des données à la demande réduit progressivement les accès à la base du portail. Elle adapte aussi le schéma de placement de chaque fragment en fonction de sa popularité et de la charge des sites qui contiennent des répliques de celui-ci. Les résultats obtenus pour l'évaluation des performances de notre solution sont satisfaisants pour le traitement de requêtes d'analyse de biodiversité et prouvent l'efficacité de notre approche d'optimisation. Nous avons également évalué la garantie du passage à l'échelle de notre solution en faisant varier le nombre de sites qui constituent l'architecture décentralisée. Les résultats obtenus pour ces expériences montrent que les performances de notre solution augmentent autant que le nombre de machines qui composent le système. Même si les résultats obtenus sont prometteurs pour la garantie du passage à l'échelle, il serait souhaitable d'évaluer les performances de notre solution sur un réseau de machines plus largement étendu. Nous envisageons d'effectuer la validation expérimentale de notre solution sur les machines de plusieurs laboratoires utilisant le GBIF.

Validation auprès du GBIF

Les entretiens effectués avec l'équipe de de GBIF France nous ont permis de mieux comprendre le fonctionnement du portail GBIF et les exigences d'interrogation et de disponibilité pour traiter les requêtes des utilisateurs. Cette collaboration nous a permis aussi de constater l'importance des dimensions taxinomiques et géographiques pour les requêtes. C'est pourquoi nous

les avons utilisées pour fragmenter les données. Récemment, nous avons présenté notre solution décentralisée non-intrusive à l'équipe du GBIF France pour une éventuelle adoption par le GBIF international. En effet, notre solution pourrait compléter les solutions actuelles des nœuds nationaux du GBIF pour mutualiser leurs ressources et améliorer efficacement le service d'interrogation des données.

8.2 Perspectives

Nos perspectives majeures visent d'une part à faciliter l'exploitation des données de biodiversité et d'autre part à généraliser notre solution afin qu'elle puisse être adoptée pour d'autres masses de données.

Intégration de données complémentaires à l'analyse de biodiversité

Notre solution a pour avantage de faciliter l'analyse des données de biodiversité en permettant aux usagers de poser des requêtes complexes. Cependant, les analyses de biodiversité nécessitent souvent des données environnementales en plus des données d'occurrences fournies par le GBIF. Par exemple, la modélisation de niche écologique est affinée avec des données abiotiques qui décrivent les paramètres physico-chimiques de la zone d'analyse. C'est pourquoi nous envisageons de compléter notre solution avec l'intégration de données environnementales. Ces dernières concernent les données climatiques et les caractéristiques chimiques tel le types de sol, la salinité, l'acidité, ... des zones géographiques. Cette perspective permettrait d'étendre notre architecture décentralisée à d'autres sources de données que le GBIF. Le travail consistera à étudier d'une part la généralisation de la stratégie de placement pour accéder à d'autres données dans le système décentralisé et d'autre part, à optimiser les performances des requêtes d'analyse complète de biodiversité.

Intégration d'outils d'analyse de biodiversité

La mise à disposition d'outils d'analyse de biodiversité dans la plateforme permettrait aux usagers d'exploiter efficacement les données de biodiversité sans se préoccuper des traitements informatique sous-jacents qui produisent les résultats des analyses. L'idée est qu'à partir d'une interface graphique, l'utilisateur puisse choisir la nature de son analyse, les taxons, la zone géographique concernée et les paramètres environnementaux et que la plateforme détermine et calcule la requête d'analyse complète correspondante. Le travail consistera donc à étudier l'adaptabilité des outils d'analyse de biodiversité existants dans le contexte de l'architecture décentralisée.

Participation du système décentralisé aux traitements intensifs d'intégration de données de biodiversité

Avec l'automatisation progressive des techniques de collecte de données d'observation de biodiversité, la quantité de données acquises augmente très rapidement. Or, tout le processus d'intégration des données est géré par le portail. Ce qui peut poser un réel problème de performances du processus d'intégration des données pour garantir le passage à l'échelle des données collectées. Si dans cette thèse nous nous sommes concentrés sur l'optimisation de l'interrogation des données de biodiversité, cette perspective permettra à notre solution décentralisée de participer au processus d'intégration. Le travail consistera donc à trouver un mécanisme de collaboration adapté entre le système décentralisé et le portail dans le processus d'intégration des données pour garantir la disponibilité et la cohérences des données sans pour autant perturber les performances du système décentralisé pour l'interrogation.

Généralisation de la solution pour d'autres masses de données

Dans cette thèse, nous avons proposé une solution d'optimisation de requêtes d'analyse de données de biodiversité. Même si la stratégie de répartition des données est axée sur la structure hiérarchique des données de biodiversité et les spécificités des analyses, les principes généraux de répartition dynamique à la demande pourraient être appliqués à d'autres masses de données. En outre, notre approche d'optimisation de requêtes dans un environnement réparti à large échelle pourrait être étendue et utilisée dans un contexte plus générique qui n'est pas lié à des types d'analyses particuliers. Cette perspective a pour objectif d'adapter dynamiquement la stratégie de répartition des données et l'approche d'optimisation de requêtes en fonction des données et des analyses. Elle devrait permettre à notre solution de supporter des données de formats différents.

Bibliographie

[Aba09] D. J. Abadi. Data management in the cloud : Limitations and opportunities. *IEEE Data Eng. Bull.*, 32(1), 2009.

[ABPH+10] A. Abouzied, K. Bajda-Pawlikowski, J. Huang, D. J. Abadi, and A. Silberschatz. Hadoopdb in action : Building real world applications. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, pages 1111–1114, 2010.

[AEADE11] D. Agrawal, A. El Abbadi, S. Das, and A. J. Elmore. Database scalability, elasticity, and autonomy in the cloud. In *Proceedings of the 16th International Conference on Database Systems for Advanced Applications - Volume Part I*, DASFAA'11, pages 2–15. Springer-Verlag, 2011.

[AKB+12] S. Agarwal, S. Kandula, N. Bruno, M. Wu, I. Stoica, and J. Zhou. Re-optimizing data-parallel computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, pages 21–21, 2012.

[BBRW09] L. Bellatreche, K. Boukhalfa, P. Richard, and K. Y. Woameno. Referential horizontal partitioning selection problem in data warehouses : Hardness study and selection algorithms. *International Journal of Data Warehousing and Mining*, 5(4) :1–23, 2009.

[BFG+08] M. Brantner, D. Florescu, D. A. Graf, D. Kossmann, and T. Kraska. Building a database on s3. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 251–264. ACM, 2008.

[Bio] BioVeL. Biodiversity virtual e-laboratory. `https://www.biovel.eu`. Online ; accessed 15-04-2015.

[BNSN12] N. Bame, H. Naacke, I. Sarr, and S Ndiaye. Architecture répartie à large échelle pour le traitement parallèle de requête de biodiversité. In *Proceeding of the 11th African Conf. on Research in Computer Science and Applied Mathematics (CARI)*, pages 143–150, 2012.

[BNSN13] N. Bame, H. Naacke, I. Sarr, and S Ndiaye. Traitement décentralisé de requêtes de biodiversité. In *Colloque National sur la Recherche en Informatique et ses Applications (CNRIA)*, page 8, 2013.

[BNSN14a] N. Bame, H. Naacke, I. Sarr, and S Ndiaye. Algorithmes de traitement de requêtes de biodiversité dans un environnement distribué. *Revue africaine de la recherche en informatique et mathématiques appliquées (ARIMA)*, 18(1) :1–18, 2014.

[BNSN14b] N. Bame, H. Naacke, I. Sarr, and S Ndiaye. Bigbio : Utiliser les techniques de gestion du big data pour les données de la biodiversité. In *Proceeding of the 12th African Conf. on Research in Computer Science and Applied Mathematics (CARI)*, pages 273–284, 2014.

[BPASP11] K. Bajda-Pawlikowski, D. J. Abadi, A. Silberschatz, and E. Paulson. Efficient processing of data warehousing queries in a split execution environment. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, pages 1165–1176, 2011.

[C+06] F. Chang et al. Bigtable : A distributed storage system for structured data. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*, OSDI '06, pages 15–15, 2006.

[Cen] Intel IT Center. Big data analytics : Intel's it manager survey on how organizations are using big data. `http://www.intel.com/content/dam/www/public/us/en/documents/reports/data-insights-peer-research-report.pdf`. Online ; accessed 15-04-2015.

[CJL+08] R. Chaiken, B. Jenkins, P. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou. SCOPE : easy and efficient parallel processing of massive data sets. *PVLDB*, 1(2) :1265–1276, 2008.

[CJP+11] C. Curino, E. Jones, R. A. Popa, N. Malviya, E.e Wu, S. Madden, H. Balakrishnan, and N. Zeldovich. Relational cloud : a database service for the cloud. In *CIDR 2011, Fifth Biennial Conference on Innovative Data Systems Research*, pages 235–240, 2011.

[DG08] J. Dean and S. Ghemawat. Mapreduce : simplified data processing on large clusters. *Commun. ACM*, 51(1) :107–113, 2008.

[dRL10] L. d'Orazio, C. Roncancio, and C. Labbé. Adaptable cache service and application to grid caching. *Concurrency and Computation : Practice and Experience*, 22(9) :1118–1137, 2010.

[ETO+11] M. Y. Eltabakh, Y. Tian, F. Özcan, R. Gemulla, A. Krettek, and J. McPherson. Cohadoop : Flexible data placement and its exploitation in hadoop. *Proc. VLDB Endow.*, 4(9) :575–585, 2011.

[F.a] Apache F. Apache activemq. `http://activemq.apache.org`. Online ; accessed 15-04-2015.

[F.b] Apache F. Apache pig. `http://pig.apache.org/`. Online ; accessed 15-04-2015.

[F.c] Apache F. Hadoop wiki :. `http://wiki.apache.org/hadoop`. Online ; accessed 15-04-2015.

[Fac] Facebook. Facebook newsroom statistics. `http://newsroom.fb.com/company-info/`. Online ; accessed 15-04-2015.

[FC] GBIF F. and GBIF C. Gbif france and canadian bif. `www.gbif.fr` and `www.cbif.gc.ca`. Online ; accessed 15-04-2015.

[FS12] Chanchary F., H. and Islam S. Data migration : Connecting databases in the cloud. In *Proceedings of the 1st International Conference on Computing and Information Technology (ICCIT)*, pages 450–455, 2012.

[GARK12] B. Gufler, N. Augsten, A. Reiser, and A. Kemper. Load balancing in mapreduce based on scalable cardinality estimates. In *IEEE 28th International Conference on Data Engineering (ICDE 2012)*, pages 522–533, 2012.

[GB] GEO-BON. Geo-bon project. `www.earthobservations.org`. Online ; accessed 15-04-2015.

[GBI] GBIF. Gbif data use case. `http://www.gbif.org/newsroom/uses`. Online ; accessed 15-04-2015.

[GGL03] S. Ghemawat, H. Gobioff, and S. Leung. The Google file system. In *ACM Symp. on Operating Systems Principles (SOSP)*, pages 29–43, 2003.

[Gop12] C. Gope. Dynamic data allocation methods in distributed database system. *American Academic and Scholarly Research Journal*, 4(6), 2012.

[GPA+13] T.C. Giannini, C.E. Pinto, A.L. Acosta, M. Taniguchi, A.M. Saraiva, and I. Alves-dos-Santos. Interactions at large spatial scale : The case of centris bees and floral oil producing plants in south america. *Ecological Modelling*, 258 :74 – 81, 2013.

[GSN09] M. Gueye, I. Sarr, and S. Ndiaye. Database replication in large scale systems : Optimizing the number of replicas. In *Proceedings of the 2009 EDBT/ICDT Workshops*, EDBT/ICDT '09, pages 3–9, 2009.

[HAD09] H. Hoffmann, A. Agarwal, and S. Devadas. Partitioning strategies for concurrent programming, 2009.

[Hob03] D. Hobern. Gbif biodiversity data architecture. Technical report, GBIF DADI, 2003.

[IBY+07] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad : Distributed data-parallel programs from sequential building blocks. *SIGOPS Oper. Syst. Rev.*, 41(3), 2007.

[IGN+12] S. Idreos, F. Groffen, N. Nes, S. Manegold, S. Mullender, and M. Kersten. Monetdb : Two decades of research in column-oriented database architectures. *IEEE Data Eng. Bull*, 2012.

[Inc] Databricks Inc. Shark, spark sql, hive on spark, and the future of sql on spark. `http://databricks.com/blog/2014/07/01/shark-spark-sql-hive-on-spark-and-the-future-of-sql-on-spark.html`. Online ; accessed 15-04-2015.

[JJR14] Li J., Naughton J., and Nehme R., V. Resource bricolage for parallel database systems. *VLDB*, 8(1), 2014.

[KGD11] S. Kamali, P. Ghodsnia, and K. Daudjee. Dynamic data allocation with replication in distributed systems. In *IEEE Intl Conf. on Performance Computing and Communications Conference (IPCCC)*, pages 1–8, 2011.

[KKAP96] Y.K. Kwok, K. Karlapalem, I. Ahmad, and N.M. Pun. Design and evaluation of data allocation algorithms for distributed database systems. *IEEE Journal on Sel. areas in Commun. Special Issue on Distributed Multimedia Systems*, 14 :1332–1348, 1996.

[Kos00] D. Kossmann. The state of the art in distributed query processing. *ACM Comput. Surv.*, 32(4) :422–469, 2000.

[KPX+11] Q. Ke, V. Prabhakaran, Y. Xie, Y. Yu, J. Wu, and J. Yang. Optimizing data partitioning for data-parallel computing. In *Proceedings of the 13th USENIX Conference on Hot Topics in Operating Systems*, HotOS'13, pages 13–13. USENIX Association, 2011.

[KTR12] L. Kolb, A. Thor, and E. Rahm. Load balancing for mapreduce-based entity resolution. In *Proceedings of the 2012 IEEE 28th International Conference on Data Engineering*, ICDE '12, pages 618–629, 2012.

[L.] Map L. Map of life. `www.mappinglife.org`. Online ; accessed 15-04-2015.

[LA00] T. Loukopoulos and Ishfaq A. Static and adaptive data replication algorithms for fast information access in large distributed systems. In *Proceeding of Intl Conf. on Distributed Computing Systems (ICDCS)*, pages 385–392. IEEE Computer Society, 2000.

[Mos] MosquitoMap. Mosquitomap project. `www.mosquitomap.org`. Online ; accessed 15-04-2015.

[NB11] R. Nehme and N. Bruno. Automated partitioning design in parallel database systems. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, 2011.

[OKH+13] M. Onizuka, H. Kato, S. Hidaka, K. Nakano, and Z. Hu. Optimization for iterative queries on mapreduce. *PVLDB*, 7(4) :241–252, 2013.

[OV11] T. M. Ozsu and P. Valduriez. *Principles of Distributed Database Systems, third edition.* Springer, 2011.

[P.a] DiGIR P. Digir protocol. `http://www.digir.net/`. Online ; accessed 15-04-2015.

[P.b] LifeWatch P. Lifewatch project. `www.lifewatch.eu`. Online ; accessed 15-04-2015.

[P.c] TAPIR P. Tapir dwg access protocol for information retrieval. `http://www.tdwg.org/activities/tapir/`. Online ; accessed 15-04-2015.

[PPR+09] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker. A comparison of approaches to large-scale data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, SIGMOD '09, pages 165–178, 2009.

[Pro] GAIA Project. Gaia and cnes centre national d'etudes spatiales. `http://smsc.cnes.fr/GAIA/Fr/index.htm`. Online ; accessed 15-04-2015.

[RF01] K. Ranganathan and I. T. Foster. Identifying dynamic replication strategies for a high-performance data grid. In *Proceedings of the Second International Workshop on Grid Computing*, GRID '01, 2001.

[Saa05] H. Saarenma. Sharing and accessing biodiversity data globally through gbif. In *ESRI User Conf*, 2005.

[SAR+14] M. A. Soliman, L. Antova, V. Raghavan, A. El-Helw, Z. Gu, E. Shen, G. C. Caragea, C. Garcia-Alvarado, F. Rahman, M. Petropoulos, F. Waas, S. Narayanan, K. Krikellas, and R. Baldwin. Orca : A modular query optimizer architecture for big data. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, pages 337–348, 2014.

[SB09] J. P. Sampoux and V. Badeau. Modélisation de la niche écologique des fétuques à feuilles fines : quels apports pour la conservation et la valorisation des ressources génétiques ? *Innovations Agronomiques, 7*, pages 79–91, 2009.

[SCSA08] G. Soundararajan, J. Chen, M. A. Sharaf, and C. Amza. Dynamic partitioning of the cache hierarchy in shared data centers. *Proc. VLDB Endow.*, 1(1), 2008.

[Sec] GBIF Secretary. Gbif data portal and gbif web site. `data.gbif.org` and `www.gbif.org`. Online ; accessed 15-04-2015.

[SJK+13] J. Schaffner, T. Januschowski, M. Kercher, T. Kraska, H. Plattner, M. J. Franklin, and D. Jacobs. Rtp : Robust tenant placement for elastic in-memory database clusters. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, pages 773–784, 2013.

[SKRC10] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. In *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, MSST '10, pages 1–10, 2010.

[SNB+14] I. Sarr, H. Naacke, N. Bame, I. Gueye, and S Ndiaye. Green and distributed architecture for managing big data of biodiversity. In *Book Chapter In Computing in Research and Development in Africa : Benefits, Trends, Challenges and Solutions*, pages 21–39. Springer International Publishing Switzerland, 2014.

[SPTB14] M. Stonebraker, A. Pavlo, R. Taft, and M. L. Brodie. Enterprise database applications and the cloud : A difficult road ahead. In *Proceedings of the 2014 IEEE International Conference on Cloud Engineering*, IC2E '14, pages 1–6, 2014.

[T+10] A. Thusoo et al. Hive - a petabyte scale data warehouse using hadoop. In *ICDE '10 : Proceedings of the 26th International Conference on Data Engineering*, pages 996–1005. IEEE, 2010.

[tE14] From the Editors. Big data and management. *Academy of Management Journal*, 57(2) :321–326, 2014.

[Tec] TechCrunch. How big is facebook's data ? 2.5 billion pieces of content and 500+ terabytes ingested every day. `http://techcrunch.com/2012/08/22/how-big-is-facebooks-data-2-5-billion-pieces-of-content-and-500-terabytes-ingested-every-day/s`. Online ; accessed 15-04-2015.

[TMX+13] M. Tu, H. Ma, L. Xiao, I. L. Yen, F. Bastani, and D. Xu. Data placement in p2p data grids considering the availability, security, access performance and load balancing. *J. Grid Comput.*, 11(1) :103–127, 2013.

[VRL04] M. Villamil, C. Roncancio, and C. Labbé. Pins : Peer-to-peer interrogation and indexing system. In *8th International Database Engineering and Applications Symposium (IDEAS 2004)*, pages 236–245, 2004.

[WC13] G. Wang and C. Chan. Multi-query optimization in mapreduce framework. *PVLDB*, 7(3) :145–156, 2013.

[Wie14] L. Wiese. Clustering-based fragmentation and data replication for flexible query answering in distributed databases. *Journal of Cloud Computing*, 3(18), 2014.

[Wor] WorldClim. Worldclim global climate data : Free climate data for ecological modeling and gis. `http://www.worldclim.org`. Online ; accessed 15-04-2015.

[XRZ+13] R. S. Xin, J. Rosen, M. Zaharia, M. J. Franklin, S. Shenker, and I. Stoica. Shark : Sql and rich analytics at scale. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD*, pages 13–24, 2013.

[ZBL12] J. Zhou, N. Bruno, and W. Lin. Advanced partitioning techniques for massively distributed computation. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 13–24, 2012.

[ZCD+12] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets : A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, 2012.