# Traitement décentralisé de requêtes de biodiversité

Ndiouma Bame[1], Hubert Naacke[2], Idrissa Sarr[1] et Samba Ndiaye[1]

[1]Département de mathématique-informatique
UCAD, Dakar SENEGAL

Prenom.nom @ucad.edu.sn

[2]LIP6

UPMC Sorbonne Universités France

Hubert.Naacke@lip6.fr

**RÉSUMÉ.** Le GBIF est un portail mondial pour le partage de bases de données dans le domaine de la biodiversité. Il a pour objectif de fédérer et partager les données de biodiversité existant dans de nombreux laboratoires à travers le monde. Avec un nombre croissant de fournisseurs qui ajoutent de nouvelles données et d'utilisateurs manifestant de nouveaux besoins qui interrogent la base, l'infrastructure est confrontée à des problèmes de disponibilité et d'expressivité limitée des requêtes. Pour faire face à ces problèmes, nous envisageons une solution qui passe à l'échelle avec un coût relativement faible. Dans cette perspective, nous proposons une architecture décentralisée et non intrusive pour interroger les données du GBIF en s'appuyant sur une infrastructure distribuée. Nous définissons une stratégie de répartition dynamique des données, adaptée au contexte du GBIF. Nous démontrons la faisabilité de notre approche par l'implémentation d'un prototype exécutant des requêtes jusqu'ici non supportées par le GBIF.

**ABSTRACT.** The GBIF is a global portal for biodiversity databases sharing. Its goal is to federate and share the biodiversity data existing in many laboratories across the world. The global biodiversity data faces two problems, namely the data availability and a poor expressiveness of queries mainly due to a growing number of users, which express more new needs. To deal with these problems, we envision a scalable and relatively low cost solution. With this in mind, we propose a non-invasive and decentralized architecture for processing GBIF queries over a distributed system. We define a dynamic strategy for data distribution that fits the GBIF requirements. We demonstrate the feasibility of our solution by a prototype implementation, which allows for processing extra query types, up to now unsupported by the GBIF portal.

**MOTS-CLÉS :** masses de données, réplication et distribution de données, cloud computing, traitement des requêtes, GBIF.

**KEYWORDS:** data replication and distribution, cloud computing, query processing, GBIF.

## 1. Introduction

Le GBIF est un système d'information qui a pour objectif de fédérer et de partager les données de la biodiversité à l'échelle mondiale [1][3][5]. Sa base de données est complétée continuellement par des correspondants nationaux. Elle contient aujourd'hui près de 400 millions d'enregistrements. Avec un nombre croissant à la fois de fournisseurs qui ajoutent de nouvelles données, et d'utilisateurs qui interrogent la base [2][5], l'infrastructure actuelle du GBIF, qui est centralisée via un portail, peine à servir toutes ces demandes en un temps raisonnable. Ces limites que nous avons recensées dans [17] posent, d'une part, un réel problème de disponibilité des données tout en empêchant un usage réellement interactif de ces données, d'autre part, les infrastructures informatiques sont en pleine évolution : les nuages informatiques (*cloud*) sont omniprésentes et permettent d'accéder à des ressources quasi-illimitées de stockage et de calcul [10]. Ceci incite à concevoir de nouvelles solutions pour la gestion de gros volumes de données, garantissant des accès rapides et un coût relativement abordable en fonction de la charge applicative [16] [8] [9]. Les objectifs de ce travail sont :
1) de décentraliser l'accès au portail à travers plusieurs participants afin de paralléliser les accès et les traitements; 2) de réduire les congestions et d'augmenter la réactivité du système vis-à-vis de ses usagers ; 3) de rapprocher les données des utilisateurs afin de réduire, le plus possible, les accès distants ; 4) de réduire le coût et/ou le temps de réponse des requêtes de l'utilisateur.
Pour ce faire, nous répliquons dynamiquement et partiellement les données du GBIF en fonction des classes d'accès des requêtes et de la localisation des utilisateurs. De fait, les données du GBIF sont répliquées partiellement sur des nœuds locaux (associant des utilisateurs à une base de données locale) en fonction des types de requêtes émises par les utilisateurs se trouvant sur un nœud local. Un fragment du GBIF est répliqué sur un nœud local à chaque fois qu'un utilisateur accède à ce fragment. Cette stratégie, qui est presque similaire au principe de cache des données en mémoire centrale, a pour avantage de décentraliser les accès aux données surtout pour les requêtes fréquentes et elle permet d'exploiter au mieux les ressources disponibles (GBIF et nœuds locaux). De plus, dès qu'un fragment est présent sur un nœud local, ce dernier est sollicité en remplacement du GBIF à chaque fois qu'une requête veut accéder au fragment. Cette solution est adaptée dans le contexte du GBIF où les correspondants nationaux ont deux atouts : d'une part, ils disposent de ressources (i.e., les nœuds locaux) pour gérer des données, et d'autre part, ils acceptent que leurs nœuds locaux participent à l'évaluation de requêtes accédant à des données situés sur d'autres nœuds locaux. Ce mécanisme de collaboration pour traiter des requêtes réparties écarte notre solution des techniques de cache de données classiques.
La suite de ce papier est organisée comme suit : la section 2 décrit notre stratégie de répartition des données dans une architecture décentralisée. La section 3 présente les mécanismes de traitement des requêtes. La section 4 détaille la validation expérimentale

de notre proposition et présente les résultats obtenus et les contributions de notre travail. La section 5 présente l'état de l'art sur le traitement de requêtes réparties.

## 2. Fragmentation et réplication dynamiques des données

Dans l'architecture proposée au [17], chaque participant $P_i$ héberge une base de données, $BD_i$ locale. Chaque $BD_i$ a le même schéma de données que la base de données du GBIF. Ceci permet aux utilisateurs connaissant le schéma de la base du GBIF d'interroger sans limitations les données du GBIF. Par contre, pour les données, seules des portions de la base du GBIF sont répliquées sur les $BD_i$. Ces dernières contiennent les données fréquemment manipulées en lecture, ce qui permet de sauvegarder la bande passante tout en réduisant le temps de réponse [19]. Pour déterminer les données à répliquer, nous nous basons sur les prédicats des requêtes qui sont soumises au participant concerné. Une telle réplication à la demande, permet de minimiser les coûts de stockage des $BD_i$ et de communication avec le GBIF tout en favorisant le parallélisme des accès et des traitements. De plus, elle permet de placer les données aux seuls endroits où elles sont utilisées. Ce type de réplication dynamique est avantageux dans un contexte où les participants accédant souvent à des données communes peuvent s'échanger leurs répliques à moindre coût.
Nous avons choisi une fragmentation horizontale dérivée. Elle est faite à la demande en utilisant les prédicats d'une ou d'un ensemble de requêtes. Lorsqu'un fragment $f$ qui existe dans un participant $P_i$, est très sollicitée par les utilisateurs $U_j$ d'un participant $P_j$, alors $f$ est répliquée une nouvelle fois depuis $P_i$ vers le participant $P_j$. Cela permettra aux utilisateurs $U_j$ d'accéder localement aux données qu'ils demandent. Cette re-réplication résulte d'une collaboration entre les participants.

## 3. Traitement des requêtes sur le GBIF

Nous décrivons l'exécution des requêtes. Le mécanisme d'exécution tient compte de la localisation des fragments nécessaires pour évaluer une requête. Nous posons les hypothèses suivantes : (i) le nombre de fragments qu'un utilisateur accède reste une petite portion de la base du GBIF ; (ii) toutes les données nécessaires à l'évaluation d'une requête tiennent dans un participant ; (iii) l'accès à un participant distant induit un surcoût (e.g., ouverture de la connexion et gestion des pannes) proportionnel au nombre de participants ; (iv) Le transfert des données ajoute un surcoût proportionnel à la quantité de données transférées. Dans ces conditions, nous évaluons une requête en favorisant au maximum les traitements locaux. Cela permet de limiter le nombre de participants au minimum nécessaire. Les avantages sont doubles : réduire le temps de consolidation du résultat final, et raréfier les situations de blocage lié à un participant inaccessible. Plus précisément, nous distinguons deux types de requêtes ; les requêtes

décomposables que l'on peut découper en un ensemble de sous-requêtes réparties. En opposition, nous avons les requêtes non-décomposables dont l'exécution nécessite que toutes les données impliquées soient transférées dans un participant.

### 3.1. Traitement de requêtes décomposables

Une requête décomposable peut être reformulée comme l'union ou l'intersection de plusieurs sous-requêtes accédant chacune à des données disjointes. Lorsque les données sont réparties sur plusieurs participants, la requête décomposable est découpée en un ensemble de sous-requêtes. Chaque sous-requête est envoyée au participant concerné. Ce mécanisme s'appuie sur le principe de MapReduce [8][12] qui consiste à paralléliser une tâche en la scindant en sous-tâches afin de minimiser le temps de réponse. Un fragment de donnée peut être répliqué à travers plusieurs participants. Le choix de la réplique à lire pour le traitement d'une requête est guidé par la minimisation des coûts de communication en favorisant les traitements locaux. Pour cela, lorsqu'un fragment $F_i$ est répliqué sur deux participants $P_i$ et $P_j$, on choisit de solliciter le participant disposant du plus grand nombre de fragments nécessaires au traitement. Lorsque $P_i$ et $P_j$ ont le même nombre de fragments impliqués dans le traitement d'une requête, alors le participant le moins chargé est choisi.

### 3.2. Traitement de requêtes non-décomposables

Une requête non décomposable ne peut pas être reformulée comme l'union ou l'intersection de plusieurs sous requêtes, car elle nécessite un traitement final plus complexe qu'une simple opération d'union ou d'intersection, par exemple un regroupement (group by) suivi d'agrégations (min, max). L'objectif visé ici est de traiter une requête quelle que soit sa complexité tout en minimisant le nombre de participants intervenant dans le traitement de la requête pour les mêmes raisons que dans le cas du traitement de requêtes décomposables (cf. section 3.1). Pour cela, on choisit les sites qui ont le plus de données nécessaires à l'exécution de la requête. Parmi ces derniers, on désigne le site disposant de la plus grande quantité de données, comme étant le site d'exécution de la requête. Les autres participants évaluent localement les éventuels prédicats de sélection de la requête, puis transfèrent ensuite leurs réponses vers le participant désigné. La requête est finalement évaluée sur le participant désigné.

## 4. Validation expérimentale

Les objectifs de la validation expérimentale sont d'évaluer la faisabilité et les performances de notre solution, puis de la comparer avec une solution de réplication totale des données du GBIF sur chaque participant sans collaboration entre participants.

## 4.1. Environnement et expériences

Nous avons utilisé l'outil de développement SimJava [18] pour émuler un cluster de 10 machines. Seule l'infrastructure physique a été émulée, la gestion des répliques et des requêtes a été implémentée réellement. Sur chaque nœud local d'un participant, le système de gestion de base de données en mémoire centrale H2 [15] est utilisé. Les communications entre les participants se font avec java et l'accès aux répliques avec jdbc. Nous avons travaillé avec une base de données miroir (dump) du GBIF. Nous avons réparti les données de sorte que chaque base contienne en moyenne 40 000 enregistrements d'occurrences d'espèces. On suppose que la taille maximale d'une base est de 100 000 enregistrements. Ce paramétrage concerne seulement les expériences avec notre solution. Pour les expériences avec la réplication totale, chaque base dispose de 500 000 enregistrements.
Nous avons mesuré le temps de réponse de chaque requête. Les requêtes testées sont composées d'opérations de sélection et de jointure (pour lier le nom de l'espèce à ses occurrences). Les temps de réponse, reportés sur les tableaux 1 et 2, correspondent à la valeur moyenne du temps mesuré pour dix exécutions successives de la même requête. On mesure le temps de réponse d'une requête entre l'instant de soumission de la requête et l'instant où l'utilisateur commence à lire le résultat.

## 4.2. Résultats et discussions

### 4.2.1. Requêtes décomposables

L'obtention des résultats escomptés pour toutes les requêtes et ceci dans des délais raisonnables avec un temps de réponse maximal de 1203 ms (tableau 1) montre la faisabilité de notre solution. Pour chaque requête, on voit nettement que le temps de réponse obtenu avec notre solution est toujours meilleur que celui obtenu avec la réplication complète. La performance de notre solution est au moins 5 fois plus rapide que la solution de réplication complète pour les requêtes décomposables. Ceci est dû au fait que les tailles des relations traitées sont plus petites avec notre solution.

### 4.2.2. Requêtes non- décomposables

Le tableau 2 montre que les résultats de notre solution sont meilleurs pour chaque requête non-décomposable donnée. En effet, aucun temps de réponse n'a atteint les 2 secondes alors que dans le cas de la réplication complète tous les temps dépassent les 6.5 secondes. Là, il importe aussi de noter que nous n'avons pas effectué la réplication de toutes les occurrences du GBIF (400 millions) mais nous avons travaillé avec une portion de 500 mille occurrences. Ce qui montre que si on avait travaillé avec la base entière du GBIF, on aurait des temps encore beaucoup plus élevés pour la solution de réplication complète. D'où l'importance d'adopter une réplication partielle.

Les résultats obtenus par notre expérimentation prouvent l'efficacité de notre solution pour sa faisabilité et pour ses performances. Ces expériences ont permis de traiter des requêtes jusqu'ici non supportées par le portail du GBIF. Ce qui permet

d'enlever la limite de l'expressivité des requêtes du GBIF actuel avec des temps de réponse acceptables. En outre notre proposition contribue à l'amélioration des performances avec la répartition des accès et la parallélisation des traitements qui réduit les problèmes liés à la congestion et la charge de travail du système central du GBIF.

| Requête | Schéma de répartition des fragments dans notre solution (P1=Site local) | Notre solution (ms) | Réplication complète (ms) |
|---|---|---|---|
| R (A) | A : P2 | 735 | 5813 |
| R (B) | B : P1 | 422 | 5828 |
| R (A, B) | A : P2 ; B : P1 | 922 | 6078 |
| R (A, C) | A : P2 ; C: P3 | 1000 | 6047 |
| R (A, B, C, D) | A : P1 ; B : P2 ; C, D : P3 | 1203 | 6109 |

*Tableau 1 : temps de réponse de requêtes décomposables*

| Requête de décompte du nombre d'occurrence | Schéma de répartition des fragments dans notre solution (P1=Site local) | Notre solution (ms) | Réplication complète (ms) |
|---|---|---|---|
| R (A) | A : P1 | 781 | 6922 |
| R (B) | B : P2 | 1000 | 6733 |
| R (A, B, C) | A, B, C : P1 | 1031 | 6891 |
| R (A, B, C) | A, B, C : P2 | 1079 | 6828 |
| R (A, B, C) | A : P1 ; B, C : P2 | 1232, | 6812 |
| R (A, B, C) | A, B : P2 ; C : P1 | 1329 | 6844 |
| R (A, B, C, D) | A, B : P2 ; C : P1 ; D : P3 | 1797 | 6906 |

*Tableau 2 : temps de réponse de requêtes non-décomposables*

## 5. Related work

Beaucoup de travaux ont concerné le traitement parallèle de requête dans les environnements distribués. En 2000, les auteurs de [22] ont fait un état de l'art du domaine qui récapitulait une bonne partie des travaux et techniques qui existaient jusqu'à cette époque. La plupart de ces techniques sont utilisés dans des technologies récentes pour améliorer l'optimisation du traitement et/ou de l'utilisation de la bande passante. Parmi ces techniques, on peut citer le *row blocking* pour optimiser l'utilisation de la bande passante lors du transfert de données, le *data shipping* ou *query shipping* pour le transfert respectif de données et de traitement. Dans notre approche, nous avons mis en place des mécanismes combinant *row blocking* et *caching* et *data shipping* pour minimiser le coût de transfert de données ainsi le *query shipping* pour exploiter au mieux le parallélisme dans notre contexte. Des travaux considérables [6][7][8][20][21] ont continué d'être menés pour aboutir à HadoopDB [13] en 2010 en passant GFS, Bigtable, HDFS, MapReduce, Hadoop, Hive, etc. Si GFS [20] et HDFS [21] sont des systèmes de gestion de fichiers adaptés à des environnements distribués, MapReduce [8] est un mécanisme de traitement parallèle de requête qui consiste à décomposer la requête en un ensemble de tâches réparties à travers les sites disposant des données concernés. Beaucoup de systèmes inspirés des principes de MapReduce ont ainsi vu le jour comme Hadoop [6]. Hadoop est une implémentation de MapReduce qui utilise le système de fichiers HDFS et offre de bonne performance pour le traitement parallèle sur de gros volume de données (plusieurs To). Cependant il n'est pas efficace pour des traitements locaux sur de petites quantités de données (quelques Mo). Hive [7][11] et HadoopDB [13] qui sont des dérivés de Hadoop, ont comme lui, les caractéristiques d'être performants avec les gros volumes de données et médiocres avec les petites

quantités de données. Hive offre une interface proche du langage SQL pour une interrogation plus aisée des données, alors HadoopDB stocke ses données dans des bases de données relationnelles avec une implémentation avec PostgreSql pour bénéficier des performances des systèmes relationnels. Il utilise Hadoop et Hive pour la gestion des communications afin de bénéficier du parallélisme. Hadoop et Hive sont fortement utilisés par Yahoo et Facebook pour la gestion de leurs quantités énormes de données (plusieurs centaines de To) [11]. Ces systèmes disposent d'une architecture Master/Slave où toutes les requêtes passent par le nœud maître appelé *Namenode*, qui stocke métadonnées sur le système global avant d'être traitées par les nœuds esclaves qui contiennent les données. La plupart de ces solutions prônent une centralisation de la coordination de l'exécution des requêtes. Cette architecture est différente de celle de notre solution où chaque nœud est à la fois maître et esclave où les métadonnées sont stockées dans un catalogue synchronisé et les données dans les bases de données locales. Ces systèmes diffèrent aussi de notre solution du fait qu'ils ne donnent de bonnes performances pour le traitement de requêtes sur une petite quantité de données alors notre solution a pour objectif de traiter toute requête quelque soit sa complexité et le volume de données impliquées dans des délais raisonnables.

## 6. Conclusion

Ce travail présente une solution de décentralisation du portail GBIF dans l'optique d'améliorer les performances liées au traitement des requêtes et de permettre aux utilisateurs de définir de nouveaux besoins sur les données. Nous avons proposé une distribution des données sur plusieurs participants qui collaborent pour améliorer les performances du système. Nous avons défini une architecture décentralisée pour traiter efficacement les requêtes des utilisateurs. Elle présente l'avantage de fonctionner sans modifier le portail GBIF existant. Nous avons proposé une stratégie de fragmentation et réplication dynamique des données et des mécanismes de traitement de requêtes dans ce contexte. Nous avons effectué les premières expérimentations de notre solution. Les résultats obtenus montrent la faisabilité de notre solution et son efficacité pour quelques requêtes typiques de l'usage en biodiversité. L'objectif principal de ces premières expériences, est de montrer la faisabilité de notre solution.

L'étude des performances notamment dans un environnement à large échelle comme le cloud constitue l'objet dans la prochaine phase de nos travaux. Pour cela, nous allons d'abord approfondir nos recherche sur les limites et les avantages de la gestion des données [10][11] dans le cloud ensuite améliorer nos mécanismes de traitement de requêtes avec une description plus formelle et des expérimentations plus complètes dans une infrastructure où les nombres de machines et de requêtes ainsi des quantités de données seront variés. L'amélioration de ces performances permettra d'adopter une solution dynamique pour passer à l'échelle avec l'augmentation des données intégrées et la croissance des usagers du GBIF.

## 7. Bibliographie

[1] D. Hobern, *GBIF Biodiversity Data Architecture, GBIF Data Access and Database Interoperability (DADI)*, 2003.

[2] GBIF, GBIF Annual Report 2009, pages 25-42, 2010.

[3] GBIF international www.gbif.org, data.gbif.org et GBIF France, www.gbif.fr.

[4] H. Saarenma. *Sharing and Accessing Biodiversity Data Globally through GBIF, ESRI User Conf.,* 2005.

[5] *The GBIF Data Portal: A practical "hands-on*" accessible au http://data.gbif.org.

[6] Apache Hadoop http://wiki.apache.org/hadoop.

[7] Hive wiki http://wiki.apache.org/hadoop/hive.

[8] J. Dean, S. Ghemawat. *MapReduce: simplified data processing on large clusters, CACM*, 2008.

[9] M. Brantner et al. Building a Database on S3, *SIGMOD*, 2008.

[10] D. J. Abadi: *Data Management in the Cloud: Limitations and Opportunities*. *IEEE Data Eng. Bull.* 32(1), 2009.

[11] A. Thusoo et al. Hive - A Petabyte Scale Data Warehouse Using Hadoop, *ICDE,* 2010.

[12] K. Bajda-Pawlikowski et al., *Efficient processing of data warehousing queries in a split execution environmen*t. *SIGMOD,* 2011:

[13] A. Abouzied et al. *HadoopDB in action: building real world applications*. *SIGMOD,* 2010.

[14] N. Bame. *Traitement de requêtes pour les données du GBIF : Utiliser un Cloud pour améliorer les performances des requêtes*, Mémoire de DEA d'Informatique, UCAD, 2011.

[15] H2 Database Engine http://www.h2database.com.

[16] D. Agrawal, S. Das, A. El Abbadi: Big data and cloud computing: current state and future opportunities. *EDBT* 2011.

[17] N. Bame, H. Naacke, I. Sarr, S Ndiaye, *Architecture répartie à large échelle pour le traitement parallèle de requête de biodiversité*, CARI'12, pp 143-150, 2012.

[18] www.simjava.com

[19] T. Loukopoulos and I. Ahmad, *Static and adaptive data replication algorithms for fast information access in large distributed systems*, ICDCS, pp 385-392, 2000

[20] S. Ghemawat et al., *The Google file system*, SIGOPS Oper. Syst. Rev. 37, 2003.

[21] D. Borthakur, *The Hadoop Distributed File System: Architecture and Design*, The Apache Software Foundation, 2007

[22] D. Kossmann et al., *The State of the Art in Distributed Query Processing*, ACM Computing Surveys, 2000, pp. 422–469.