Algorithmes de traitement de requêtes de biodiversité dans un environnement distribué

Ndiouma Bame, Hubert Naacke, Idrissa Sarr, Samba Ndiaye

▶ **To cite this version:**

Ndiouma Bame, Hubert Naacke, Idrissa Sarr, Samba Ndiaye. Algorithmes de traitement de requêtes de biodiversité dans un environnement distribué. Revue Africaine de Recherche en Informatique et Mathématiques Appliquées, 2014, Volume 18, 2014, pp.1-18. 10.46298/arima.1976 . hal-01300092

HAL Id: hal-01300092

https://inria.hal.science/hal-01300092

Submitted on 8 Apr 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Algorithmes de traitement de requêtes de biodiversité dans un environnement distribué

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

[1] Département de mathématique-informatique
Université Cheikh Anta Diop
Dakar, SENEGAL
prenom.nom@ucad.edu.sn

[2] Sorbonne Universités, UPMC Univ Paris 06
LIP6 Laboratory
Paris, France
prenom.nom@lip6.fr

RÉSUMÉ. Le portail du GBIF contient une description de la plupart des collections de données de la biodiversité mondiale. Il est confronté à des problèmes de disponibilité et d'expressivité limitée des requêtes liés à un nombre d'utilisateurs grandissant et manifestant sans cesse de nouveaux besoins. 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 nous appuyant sur une infrastructure de type Cloud. Nous définissons une stratégie de répartition dynamique des données et des algorithmes de traitement de requêtes, adaptés au contexte du GBIF. Nous démontrons la faisabilité et l'efficacité de notre approche par l'implémentation d'un prototype exécutant des requêtes jusqu'ici non supportées par le GBIF.

ABSTRACT. The GBIF portal contains a description of most of the global biodiversity data. It faces two problems, namely the data availability and a poor expressiveness of queries, mainly due to a growing number of users which keep expressing new needs. To tackle 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 cloud infrastructure. We define a dynamic strategy for data distribution and queries processing algorithms that fit the GBIF requirements. We demonstrate the feasibility and efficiency 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, nuage informatique, cloud computing, traitement des requêtes, GBIF.

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

Received, March 31, 2013
Revised, December 31, 2013
Accepted, January 7, 2014

ARIMA Journal, vol. 18 (2014), pp. 1-18.

1. Introduction

Le GBIF est un consortium international visant à fédérer et partager les données de biodiversité à l'échelle mondiale [20, 33]. Il est reconnu comme étant la référence, pour les données primaires de biodiversité, sur laquelle s'appuient les autres initiatives (e.g., LifeWatch, GEOBON). Sa base de données est complétée continuellement par les correspondants nationaux du consortium. Elle contient aujourd'hui plus de 400 millions d'enregistrements [33]. Sa taille croit continuellement, et atteindra prochainement plusieurs téraoctets. Avec un nombre croissant de fournisseurs qui ajoutent de nouvelles données et d'utilisateurs qui interrogent la base, l'infrastructure actuelle peine à servir toutes les demandes en un temps raisonnable. Ce qui pose un réel problème de disponibilité des données tout en empêchant un usage réellement interactif des données du GBIF. D'autre part, les infrastructures informatiques sont en pleine évolution : les nuages informatique (cloud) sont omniprésents et permettent d'accéder à des ressources quasi-illimitées de stockage et de calcul [1]. Ceci incite à concevoir de nouvelles solutions pour la gestion de gros volumes de données [13, 19, 2], garantissant des accès rapides et un coût relativement abordable en fonction de la charge applicative [4, 12, 9]. De plus, le GBIF ne supporte, à ce jour, que le traitement de quelques requêtes prédéfinies, bien que les usagers ne cessent d'exprimer de nouveaux besoins pour d'autres types de requêtes plus complexes. Dans [7], nous avons décrit l'architecture du portail du GBIF et identifié les problèmes liés à son fonctionnement. Le premier objectif de cet article est d'augmenter l'expressivité des requêtes proposées aux utilisateurs. Le deuxième objectif est d'apporter plus de disponibilité au GBIF, en proposant une solution garantissant qu'une requête est traitée efficacement bien que le nombre et la complexité des requêtes augmentent.

Pour ce faire, nous partons du constat que les utilisateurs du GBIF disposent de ressources propres pouvant servir à gérer des données. Par exemple, ils peuvent utiliser leur propre machine, celles du correspondant national du GBIF, ou plus généralement des ressources de stockage et de calcul dans un cloud. De ce fait, nous exploitons ces ressources pour répondre aux besoins de gestion de données des utilisateurs. Nous proposons une gestion collective et coordonnée des ressources des différents utilisateurs afin d'offrir à chacun un service de traitement de requêtes qui utilise plusieurs ressources lorsque cela s'avère plus efficace. Ainsi l'efficacité est accrue en comparaison avec une solution où chaque utilisateur exploiterait exclusivement ses propres ressources.

Par la suite nous appelons nœuds locaux les ressources propres des utilisateurs. Pour exploiter les nœuds locaux, notre méthode consiste (i) à disposer d'un système de gestion de données sur chaque nœud local, et (ii) à répliquer dynamiquement et partiellement les données du GBIF en fonction des requêtes. Une requête accède à une partie de la base du GBIF, formée de plusieurs fragments. Un fragment du GBIF est créé et répliqué sur un nœud local à chaque fois qu'un utilisateur accède à ce fragment et que ce dernier ne se trouve pas déjà dans une base de données locale. Le principe de cette méthode de réplication, ressemble à celui des mécanismes de cache des données en mémoire. Cette méthode présente l'avantage de décentraliser les accès aux données surtout pour les requêtes fréquentes, et d'exploiter au mieux les ressources disponibles. Lorsqu'une requête doit accéder à un fragment, si la réplique locale du fragment existe, alors elle est utilisée en priorité. Cela évite d'accéder au fragment de référence sur le site du GBIF. Cette solution est adaptée au contexte du GBIF où les données sont rarement modifiées et donc les répliques locales correspondent le plus souvent à celles du GBIF. De plus, les nœuds

locaux peuvent collaborer pour exécuter une requête manipulant des données se trouvant sur plusieurs nœuds locaux. Ce mécanisme de collaboration pour traiter des requêtes réparties différencie notre solution des techniques de cache de données classiques [29, 3]. Dans le cas de requêtes complexes devant traiter de nombreux fragments, notre solution permet de paralléliser les opérations d'une requête qui portent sur des fragments disjoints. Cela apporte un gain en performance significatif.

Nous résumons les principales contributions de notre travail :

1) L'augmentation de l'expressivité des requêtes par la proposition d'un système où l'utilisateur peut poser ses requêtes sans limitation. Cela élargit le champ des usages pour l'analyse des données de biodiversité.

2) La proposition d'une stratégie de réplication et de répartition de données à la demande pour optimiser leur accès. La stratégie s'avère efficace quel que soit le placement initial des données sur un seul ou plusieurs sites.

3) La conception d'un algorithme d'optimisation de requêtes qui exploite le parallélisme pour minimiser le temps de réponse. L'algorithme tient compte des coûts de transfert et de traitement afin de déterminer l'exécution optimale d'une requête.

4) L'implémentation et la validation de notre approche en deux phases : une validation qualitative auprès de l'équipe du GBIF France, et une validation quantitative, par simulation, pour mesurer le gain de performance du traitement des requêtes.

L'article est organisé comme suit : la section 2 présente l'état de l'art. La section 3 décrit l'architecture que nous proposons pour décentraliser l'accès aux données de biodiversité. La section 4 montre notre stratégie de fragmentation et réplication des données à la demande. Dans la section 5, nous définissons des algorithmes pour le traitement efficace des requêtes en maximisant le parallélisme. Nous validons expérimentalement notre approche en section 6 de répartition des données et de traitement des requêtes. La section 7 présente les conclusions et les perspectives.

2. Etat de l'art

2.1. Exploitation des données de biodiversité

Plusieurs travaux [33, 14, 27, 28] visant une meilleure exploitation des données de biodiversité ont été menés. Mais les solutions proposées sont limitées parce qu'elles correspondent soit à des portails thématiques dédiés [27] soit à des portails nationaux [14]. Une de ces solutions, MostiquoMap [27], permet de lancer des requêtes via une interface graphique et de visualiser les résultats sous forme de carte mais concerne uniquement les moustiques. Brièvement, ces solutions ne répondent donc pas aux objectifs du GBIF qui veut rendre accessibles les données de la biodiversité, sur tous les thèmes et tous les pays [20, 33]. Notre solution surmonte ces limitations en traitant les données indépendamment de leur localisation et de leur thème. Notre solution permet d'interroger l'ensemble des données primaires de biodiversité. Elle permet aussi de traiter des requêtes complexes et coûteuses (calcul des co-occurrences de deux ou plusieurs espèces dans une région). Enfin, elle supporte l'intégration de nouvelles fonctionnalités notamment l'ajout d'outils d'analyse.

2.2. Répartition des données à la demande

Les performances des systèmes de gestion des bases de données distribuées dépendent fortement de la répartition des données. Cette répartition s'appuie sur la fragmentation et la réplication des données. La plupart des stratégies de répartition [26, 5, 24] sont statiques car conçues au moment de la conception de la base de données répartie. Elles s'adaptent mal aux systèmes dynamiques dont la charge fluctue fortement. C'est pourquoi des mécanismes de distribution dynamique de données ont émergé [26, 22, 17]. Loukopoulos *et al.* ont proposé un algorithme adaptatif d'allocation de données dans le contexte dynamique du Web. Néanmoins, cette méthode doit supposer l'existence des fragments primaires déjà dans les sites. Plus récemment, Gope [17] a proposé une technique d'allocation dynamique de données basée sur un système de non-réplication. Ce système réalloue les données en respectant les changements des motifs d'accès aux données avec la contrainte temporelle. Mais ici aussi, la préexistence des fragments dans les sites doit être supposée. De plus, l'absence de réplication des données peut être pénalisante dans les nombreux cas où des traitements parallèles seraient possibles. Notre approche définit les fragments en fonction des prédicats des requêtes ; ensuite elle place ces fragments dans les mêmes sites que les requêtes qui les sollicitent et enfin, elle peut répliquer un fragment au cours du traitement de la requête. Cette stratégie de réplication de fragment au cours du traitement s'adapte aux différentes capacités en bande passante des réseaux.

2.3. Optimisation du traitement distribué de requête

Le traitement de requêtes dans les environnements distribués est un domaine toujours actuel et très étudié [12, 13, 19, 9, 1, 6, 4, 35]. En 2000, Kossmann et al. [23] ont réalisé un état de l'art très complet du domaine. L'apparition de gros volumes de données Web et la question de leur traitement a introduit de nouvelles problématiques. Les travaux de Google, GFS [16], BigTable [10], MapReduce [12] sont des réponses à ces problématiques. D'autres travaux ont suivi, notamment Hadoop [13]. Celui-ci est une implémentation open source de MapReduce. Hadoop est un système de traitement de données évolutif pour le stockage et le traitement par lot de gros volumes de données. Il s'appuie sur le système de gestion de fichiers distribué HDFS [8]. Beaucoup de briques logicielles ont été posées au dessus de Hadoop. D'abord Hive [19, 34] qui est une infrastructure de datawarehouse utilisant le système de fichiers HDFS. Il propose un langage de requête HiveQL proche de SQL. Notons que Hive tout comme Hadoop sont jugés lents du fait de leurs mécanismes d'exécution de requêtes qui stockent les résultats intermédiaires dans le système de fichiers (HDFS ou GFS), ce qui génère de nombreuses entrées/sorties sur disque. Ainsi, pour bénéficier des performances des SGBD relationnels en termes d'optimisation locale des traitements, HadoopDB [6, 2] a vu le jour. HadoopDB stocke ses données dans des bases de données relationnelles PostgreSQL et utilise Hadoop et Hive pour la gestion des communications afin de bénéficier du parallélisme offert par ces systèmes. Mais HadoopDB non plus n'excelle pas dans les traitements locaux sur de petites quantités de données. Cette limitation est levée par Shark [35], une solution plus récente, basée sur Hadoop et Hive, et qui utilise les fonctionnalités des RDDs (pour Resilent Distributed Datasets) pour charger et partager les données en mémoire. Ses performances selon [35], sont 40 à 100 fois meilleures que Hadoop. Cependant, Shark partitionne et alloue les données comme Hadoop, Hive et HadoopDB, avec une fonction de hachage globale. Ceci ne permet pas à Shark de s'adapter aux contextes où la distribution se fait à la demande et où on ne connait pas à priori l'emplacement futur d'un fragment. En définitive, toutes

ces solutions ne s'adaptent pas bien aux cas d'usage des données de biodiversité où un traitement peut porter sur une quantité limitée de données tout étant coûteux.

D'autres mécanismes d'optimisation de requêtes dans ces environnements dits aussi "large échelle" ont été proposées [36, 25, 30, 11]. Mais tous ces mécanismes se focalisent uniquement sur un type de traitement spécifique [11, 25] et ne s'adaptent donc pas à un contexte plus général. Pour remédier à cela, la solution ESQP [36] pousse les traitements vers les sites qui disposent des données tout en tenant compte de leur charge pour sélectionner les répliques à traiter. ESQP suppose un retour asynchrone des résultats à l'utilisateur et découpe une requête en un ensemble de sous-requêtes en fonction des fragments nécessaires. Chaque sous-requête est dupliquée, d'une part, autant de fois que le fragment à traiter dispose de répliques et d'autre part, en fonction de la localisation de la réplique. La sous-requête sera alors traitée sur le premier site disponible (libre). Mais cette solution ne considère ni les cas où des traitements supplémentaires doivent être effectués sur les résultats issus des sous-requêtes, ni les cas où le délai d'attente d'un site disposant d'une réplique est trop long au moment où d'autres sites voisins sont libres. Beaucoup de travaux continuent de d'être menés dans le domaine de l'optimisation des requêtes dans les architectures décentralisées. Une partie de ces travaux se consacre aux traitements transactionnels qui mettent fortement l'accent sur la cohérence des données. Une autre partie de ces travaux se concentrent sur des traitements analytiques spécifiques. Toutes ces solutions proposées, parce qu'elles apportent un surcoût de traitement ou possèdent des fonctionnalités limitées, ne s'adaptent pas au contexte de l'interrogation des données de biodiversité où les traitements nécessitent des opérations complexes (*ie.* de nombreuses jointures et agrégations).

3. Architecture décentralisée et composants

Pour résoudre les problèmes de fonctionnalités et d'optimisation liés au fonctionnement et à l'architecture du portail GBIF que nous avons identifiés et détaillés dans [7], nous proposons une solution pour répartir le traitement des requêtes et les données interrogées sur plusieurs machines afin d'allouer plus de ressources de stockage et de calcul. L'objectif étant de satisfaire les demandes croissantes des utilisateurs tout en évitant la concentration des demandes en un seul nœud qui est une source de congestion. Cette décentralisation des accès permet en outre de paralléliser les traitements. La figure suivante décrit l'architecture proposée où les principaux composants sont :

– **Le portail GBIF :** c'est un entrepôt récapitulant l'ensemble des données primaires de biodiversité accessibles à travers le réseau GBIF [20]. Cette base utilisée pour traiter directement les requêtes ou servir de passerelle pour accéder aux bases des fournisseurs. On accède aux données de la base du portail par navigateur web et/ou par des appels web service [33][14, 31, 32].

– **Site S_i :** Les différents sites locaux sont reliés par un réseau et collaborent pour échanger des données et traiter des requêtes, grâce au service de requêtes. Un utilisateur accède aux données du GBIF à travers un site S_i. Chaque site, S_i, héberge une base de données locale BD_i, un service de requête SR et un catalogue (non représentés sur la figure) :

- **La base de données locale BD_i** est vide au démarrage et est alimentée au fur et à mesure que de nouvelles requêtes demandant de nouvelles données sont soumises pour

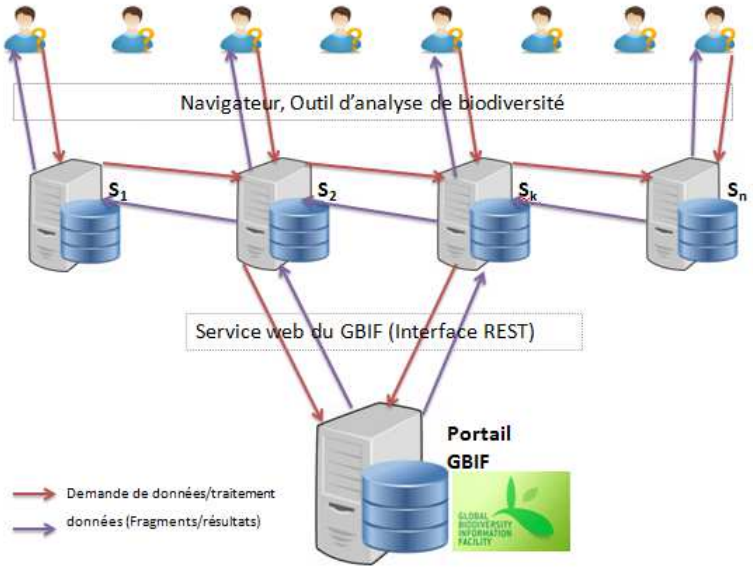

Figure 1. *Architecture distribuée pour traiter les requêtes du GBIF*

leurs traitements.

- **Catalogue :** chaque site dispose d'un catalogue local qui contient toutes les informations relatives à la localisation de chaque fragment et de ses répliques à travers le système. Composant essentiel, le catalogue, grâce à son contenu de localisation des données permet de calculer le plan d'exécution d'une requête.

- **Service de requêtes SR :** c'est une couche au dessus des BD locales (BD_i) des sites S_i, qui gère les communications et collaborations entre les différents sites. Il sert d'interface entre le système, les utilisateurs et le portail GBIF. A la réception d'une requête utilisateur, il effectue l'analyse de cette dernière pour déterminer l'ensemble des fragments de données nécessaires au traitement de la requête. Ensuite, il consulte le catalogue local pour déterminer la localisation de tous les fragments de données impliquées et celle de leurs répliques si elles existent. Le service de requête effectue l'optimisation des requêtes et coordonne leurs traitements. Si la totalité ou une partie des données nécessaires à l'évaluation de la requête n'est pas disponible dans les bases locales, le service de requêtes invoque le portail pour obtenir les données. Par ailleurs, il est chargé de retourner les résultats issus de l'exécution des requêtes aux utilisateurs.

4. Fragmentation et réplication des données à la demande

Nous proposons de répliquer des portions de la base du GBIF sur les bases locales (i.e. les BD_i). Les portions répliquées contiennent les données fréquemment lues par les requêtes, dans le but de sauvegarder de la bande passante tout en réduisant le temps de réponse. Par la suite, nous décrivons la méthode pour définir quelles portions sont répliquées, et comment elles peuvent relayer la base du GBIF. Le portail du GBIF offre une interface multicritères pour poser des requêtes. Un critère porte sur un attribut ; il pré-

cise la ou les valeurs recherchées pour cet attribut. Plus formellement, une requête est définie comme un ensemble de prédicats ayant la forme « attribut $\in V$ » avec V étant un ensemble fini de valeurs. Le résultat de la requête contient seulement les occurrences d'observation qui satisfont tous les prédicats. Ainsi, nous pouvons caractériser, au moyen d'une conjonction de prédicats, le résultat d'une requête évaluée par le portail. Ce résultat constitue un fragment stocké dans la base locale du site. Notons que ce mode de fragmentation à la demande, en fonction des requêtes, n'aboutit pas des fragments disjoints contrairement à la fragmentation horizontale employée dans les systèmes de gestion de données parallèles. En réalité, une occurrence d'observation peut être contenue (répliquée) dans plusieurs fragments, de manière redondante. Pour cette raison nous appelons notre approche fragmentation et réplication à la demande.

Notre approche consiste à réutiliser les données fournies par le portail du GBIF afin d'évaluer ultérieurement d'autres requêtes, tout en évitant de re-solliciter le portail. Lorsqu'un utilisateur pose une nouvelle requête R, nous devons déterminer si les données obtenues lors des invocations précédentes du portail suffisent pour évaluer R. Pour cela nous devons comparer R avec les fragments créés lors des requêtes précédentes. Plus précisément, à partir de l'expression algébrique de R (dans l'algèbre relationnel) nous reformulons R pour qu'elle imbrique une sélection (notée S) dont le prédicat est noté p. S'il existe un ensemble de fragments qui «contient» S, alors il est possible d'évaluer R entièrement à partir de cet ensemble. Le test d'inclusion entre S et les fragments s'effectue en comparant les prédicats des fragments avec p (voir les techniques existantes de réécriture de requête sur des vues [18]).

Bien que le portail du GBIF offre une interface d'accès à une seule relation (i.e. les occurrences d'observation). La fragmentation et la réplication à la demande que nous proposons peut être généralisée pour un service offrant l'accès à plusieurs relations formant un schéma en étoile, comme expliqué ci-après. Le schéma relationnel du GBIF est constitué d'une table principale de très grande taille (la table des occurrences d'observation d'espèces contenant plusieurs centaines de millions de nuplets). Le schéma est en étoile : les autres tables dites annexes sont reliées par une association de type 1-N avec la table principale. Une contrainte d'intégrité référentielle existe entre la clé d'une table annexe et un attribut de type clé étrangère dans la table principale. Par exemple, le nom canonique d'une espèce est référencé dans la table principale et détaillé dans la table annexe contenant tous les noms canoniques. Les tables annexes ont une taille beaucoup plus petite que la table principale. Ainsi, les tables annexes pourraient être entièrement répliquées, à moindre coût, dans les bases locales. Cela permettrait d'évaluer localement une requête dont le prédicat porte sur un attribut d'une table annexe. Le schéma en étoile serait ainsi reflété dans les bases locales, avec l'avantage d'être moins redondant qu'un schéma dénormalisé en une seule relation.

Afin d'améliorer la localité des accès, le résultat d'une requête posée par S_i sur le portail est stocké dans la base locale BD_i. Dans le cas, rare, où au moins deux sites accèdent simultanément au portail pour demander le même fragment (ou plus généralement un fragment inclus dans l'autre), un mécanisme de coordination permet de factoriser l'accès au portail : seul un site accède au portail, puis met à disposition le fragment pour le deuxième site.

De plus, des réplications de fragments d'un site local vers un autre peuvent se produire au cours de l'exécution d'une requête, lorsque cela permet de maximiser le parallélisme

des requêtes coûteuses. Ce qui favorise une réplication adaptative des données pour optimiser le traitement des requêtes sur les groupes de fragments invoqués en même temps.

Finalement, pour pouvoir stocker les fragments, les bases locales ont toutes un schéma identique à celui de la base GBIF. Cela permet à tout utilisateur connaissant le schéma de la base du GBIF d'interroger sans limitation les données du GBIF, notamment de poser des requêtes non supportées par le portail.

5. Algorithmes de traitement de requête de biodiversité

5.1. Modèle de requête

Une requête d'un utilisateur est caractérisée par un ensemble de fragments F et un ensemble d'opérations. Parmi ces dernières, on distingue l'ensemble des opérations parallélisables que nous notons OL et l'ensemble des opérations globales, noté OG, à appliquer sur l'ensemble des résultats de OL sur F. Par exemple dans la requête de calcul des co-occurrences de plantes et d'abeilles avec un maillage de 100 mètres, F est constitué des fragments de toutes les plantes et de toutes les abeilles dans la zone de l'étude, les opérations OL consistent à calculer les densités des plantes et des abeilles dans chaque maille et les opérations OG sont une superposition des densité des plantes et des abeilles dans chaque maille. Ainsi nous notons une requête par R (F, OL, OG) et considérons que le coût et la quantité de données de résultat d'un traitement OL est proportionnel au nombre de fragments et est beaucoup plus important que le coût de OG et le coût de transfert de ses résultats partiels. En effet, pour calculer la densité ou l'abondance d'une espèce dans chaque maille ou pavé de la zone d'étude, il faut d'abord créer les pavés et identifier pour chaque occurrence de l'espèce son pavé d'appartenance. Ensuite, il faut agréger les occurrences de l'espèce dans chaque maille pour y calculer sa densité. Ces traitements qui aboutissent aux densités de chaque espèce de l'étude dans chaque maille de la zone d'étude, peuvent se faire en parallèle sur les occurrences des différentes espèces, et sont largement plus coûteux que le calcul final de la co-occurrence qui consiste à prendre des décisions selon des règles établies sur les densités. Ces calculs de densité coûteux sont très fréquents dans l'analyse des données de biodiversité. En effet, les études [15] de migration, de distribution, la modélisation de niche écologique, les relations de proie/prédation font intervenir des calculs de densités ou d'abondance d'espèce.

5.2. Traitement de requête avec accès au GBIF

Dans cette sous-section, nous présentons la manière dont les requêtes sont traitées lorsque les données ne sont pas toutes disponibles dans les sites locaux. Lorsqu'une requête doit accéder à des données disponibles seulement depuis le portail du GBIF, alors ces dernières sont lues du GBIF puis transférées vers les sites locaux où elles seront insérées. Les sites locaux recevant les données du GBIF sont désignés de telle sorte que chaque site reçoive au plus un fragment (en supposant que le nombre de sites locaux est supérieur au nombre de fragments lus par une requête). Cela permet de maximiser le parallélisme lors de l'insertion des nouveaux fragments et lors des opérations locales (OL) sur chaque fragment.

5.2.1. Exécution d'une requête

La figure ci-dessous présente les étape du traitement d'un requête. A la réception d'une requête, le SR détermine les fragments nécessaires à son exécution. Lorsqu'il existe des fragments impliqués et qui sont disponibles uniquement au GBIF (F_G), alors le SR formule les appels web service correspondants ($WSR(F_G)$) qu'il envoie au GBIF pour lire ces fragments. Ces fragments sont transférés vers le site demandeur qui redirige chacun d'eux vers un site local (S_j) différent des placements des autres fragments de la requête où ils seront insérés en parallèle ($Insert(F_G^j)$). Quand tous les fragments sont disponibles (après les insertions des nouveaux fragments), le site qui a reçu la requête d'utilisateur coordonne son traitement en formulant les sous-requêtes ($R(F^j, OL)$) en fonction de la localisation de chaque fragment impliqué et des traitements OL de la requête. Chaque sous-requête est envoyée au site concerné qui la traite et retourne le résultat (f^j) au coordinateur. Le coordinateur consolide tous les résultats partiels des traitements OL (f) et y applique les traitements OG puis envoie le résultat à l'utilisateur ($R(f, OG)$). La figure suivante illustre le mécanisme d'exécution d'une requête qui nécessite des données disponibles uniquement au GBIF.

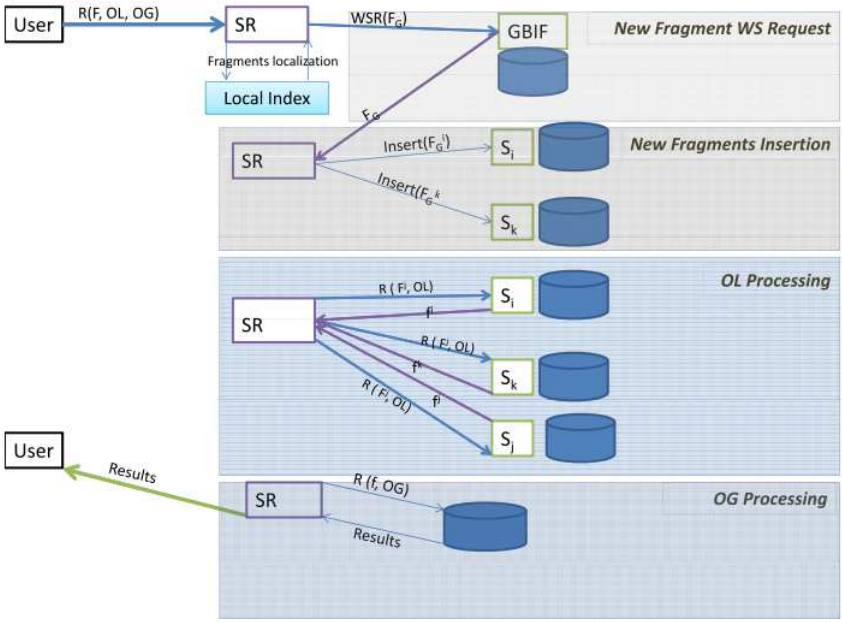

Figure 2. *Traitement de requête avec accès au GBIF*

5.2.2. Estimation du coût d'une requête

Le coût d'une requête dépend des transferts des données à lire du GBIF, de leurs transferts et insertions dans les bases locales, du coût des traitements OL, des transferts de résultats partiels vers le site coordinateur et du coût des traitements OG. Nous notons :
T_G : le temps de sélection au GBIF et de transfert d'un fragment du GBIF vers un site local,

T_I : le temps de transfert dans le réseau local et d'insertion d'un fragment dans un site,
T_{OL} : le temps d'un traitement local OL sur un fragment,
T_p : le temps de transfert d'un résultat partiel vers le coordinateur,
T_{OG} : le temps d'un traitement global OL sur un fragment,
F_G est l'ensemble des fragments disponibles uniquement au GBIF et nécessaire à la requête.
En supposant que tous les fragments ont la même taille et qu'une requête R (F, OL, OG) a pour coût C_R, alors nous avons :

$$C_R = |F_G| * T_G + T_I + T_{OL} + T_p + |F| * T_{OG}$$

Le temps de transfert à partir du GBIF et le temps des traitements OG dépendent respectivement du nombre de fragments disponible uniquement au GBIF et du nombre de fragments impliqués par la requête. Les insertions des nouveaux données s'imposent puisqu'elles n'étaient pas encore disponibles dans les sites locaux et les transferts permettent de paralléliser ces insertions et de répartir les données à travers les sites. Ce qui favorise le parallélisme des prochaines requêtes. Cette approche permet aussi de traiter des requêtes n'ayant pas besoins d'accéder au GBIF ($|F_G| = 0$) en parallélisant au maximum la requête, lorsque le nombre de fragments d'une requête est inférieur au nombre de sites du système.

5.3. Traitement de requête dans les sites locaux

Nous considérons, maintenant que toutes les données qu'une requête accède, existent parmi les sites locaux. Autrement dit, une requête est traitée sans aucun accès au GBIF. Nous faisons face à un problème de traitement de requête dans un environnement distribué où les données sont réparties et partiellement répliquées. L'objectif étant de minimiser le temps de réponse des requêtes en parallélisant les traitements les plus coûteux, l'option de transférer ou non des fragments vers des sites libres peut impacter le temps de réponse. Nous explorons deux stratégies de traitement des requêtes :

– Une stratégie, que nous appelons ParaMax, maximise le parallélisme de telle sorte que chaque fragment de la requête soit traité par site local différent. Cette stratégie génère des transferts de fragments afin de traiter une requête.

– Une stratégie, que nous appelons EcoTrans, consiste à impliquer seulement les sites disposant déjà d'au moins un fragment de la requête. Cette stratégie traite une requête sans transférer aucun fragment.

Puis nous présentons une approche pour déterminer quelle stratégie choisir, pour chaque requête, afin d'optimiser le temps de traitement des requêtes. Nous approfondissons notre étude en explorant le bénéfice, à long terme, de la stratégie Paramax.

5.3.1. Stratégie ParaMax

L'objectif étant de minimiser le temps de réponse d'une requête, le parallélisme maximum où chaque site traite le minimum de données possible pour les traitements les plus coûteux peut être avantageux. Ainsi, cette stratégie parallélise les traitements locaux sur l'ensemble des sites de sorte que chaque site traite un fragment. Pour cela, à chaque site qui héberge une ou plusieurs répliques de fragments impliqués dans la requête, est attribué un fragment qu'il stocke et qui n'est pas encore accordé (attribué) à un autre site. Les fragments restants de la requête sont ensuite attribués un à un aux sites libres, sachant qu'un seul fragment de la requête est attribué à un site. De cette façon, les sites traiteront en parallèle leurs fragments (un pour chaque site) et enverront leurs résultats au site

coordinateur du traitement de la requête (site où la requête a été soumise) qui effectue les traitements supplémentaires pour retourner le résultat final à l'utilisateur. La figure suivante illustre le principe de traitement de requête de cette approche.

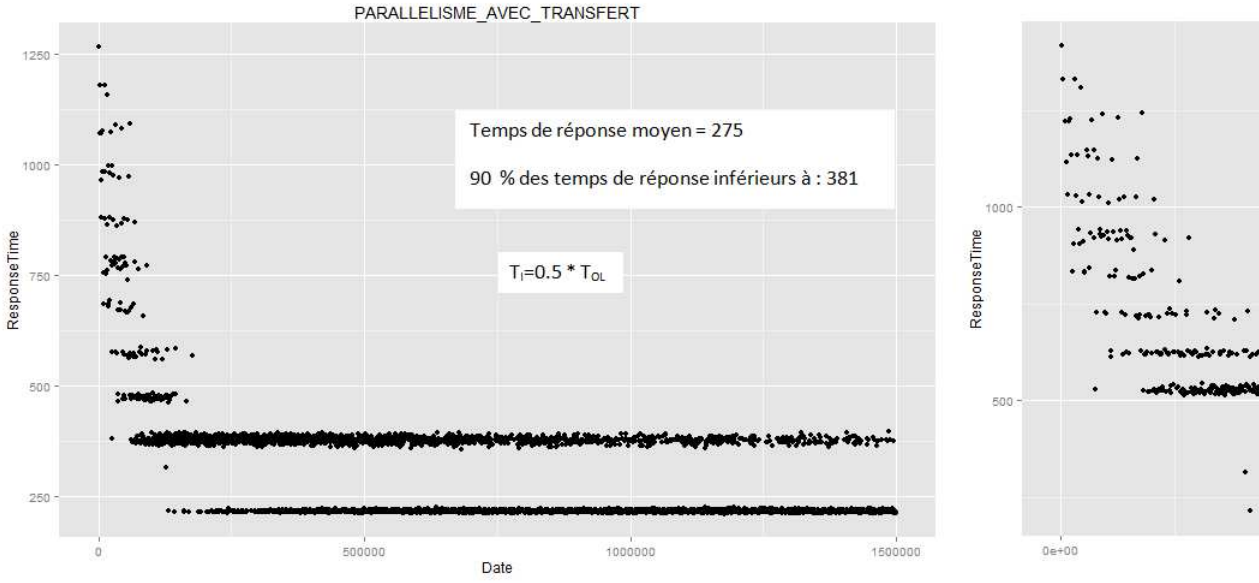

Figure 3. *parallélisme maximal avec possibilité de transfert de fragments*

Cette stratégie qui parallélise les transferts des fragments avant leurs traitements, offre de bons temps de réponse lorsque le coût de transfert de fragment est négligeable devant le coût des traitements locaux. Cependant, lorsque ce transfert n'est pas négligeable ou est supérieur au coût de traitement local, alors il peut pénaliser le temps de réponse de la requête. Le coût d'une requête avec les mêmes notations utilisées en section 5.2, est :

$$C_R = T_I + T_{OL} + T_p + |F| * T_{OG}$$

5.3.2. Stratégie EcoTrans

Le parallélisme maximum vu dans la sous-section précédente, peut réduire le temps de réponse d'une requête. Cependant, lorsque les coûts de transfert sont élevés, il peut au contraire l'augmenter. Ainsi, nous étudions une solution qui maximise le parallélisme des traitements coûteux à travers les sites qui disposent de fragment impliqués dans la requête. L'objectif étant de minimiser le temps de réponse de la requête, cette stratégie parallélise les traitements les plus coûteux en évitant les transferts des fragments avant leurs traitements. Elle vise donc à répartir les traitements OL travers les sites qui dispose une ou plusieurs répliques de fragments impliqués dans la requête. En effet, en absence de transfert de fragment avant les traitements OL, le temps de réponse dépend principalement des OL (car on suppose que $OL >> OG$). Ainsi, l'objectif revient à minimiser les OL. Plus précisément, les OL étant traités en parallèle, l'objectif consiste à minimiser la durée de l'OL le plus long, donc à minimiser le nombre de fragments sur le site participant qui

traite le plus grand nombre de fragments. Finalement, pour de réduire le coût de transferts des résultats des traitements OL, le site qui a le plus de fragments à traiter est choisi pour être coordinateur du traitement de la requête. La figure ci-dessous détaille les principes de traitements de cette stratégie.

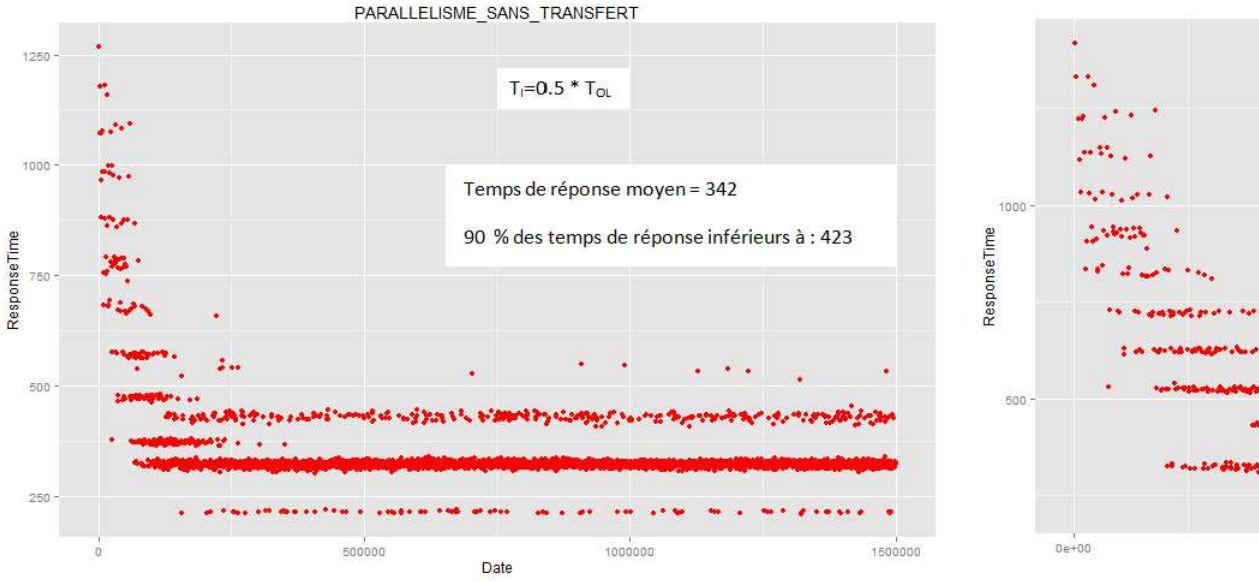

Figure 4. *Parallélisme maximal sans transfert de données*

Cette stratégie offre de bons temps de réponse lorsque les fragments sont très répliqués à travers les sites et peut aboutir à un parallélisme maximum où un site traite un seul fragment. Son coût dépend principalement du nombre de fragments traités au coordinateur, du plus grand nombre de fragments traités dans un seul site qui n'est pas coordinateur et n'a aucune dépendance du coût transfert d'un fragment. Nous notons :
T_{OL} : le temps d'un traitement local OL sur un fragment,
T_p : le temps de transfert d'un résultat partiel vers le coordinateur,
T_{OG} : le temps d'un traitement global OL sur un fragment,
F^i : l'ensemble des fragments à traiter au site coordinateur S_i,
F^j : l'ensemble des fragments traités au site S_j avec S_j (avec $S_j \neq S_i$).
En supposant que tous les fragments ont la même taille, le coût C_R d'une requête $R(F, OL, OG)$ est :

$$C_R = Max[Max(|F_j| * (T_{OL} + T_p)), |F_i| * T_{OL}] + |F| * T_{OG}$$

5.3.3. Stratégie de choix entre ParaMax et EcoTrans

Les deux stratégies que nous venons d'étudier ne sont pas optimales dans tous les cas qui peuvent se présenter. ParaMax est seulement bénéfique lorsque le coût de transfert est petit par rapport au coût d'un traitement OL, même si elle converge vers une situation où les transferts deviennent de plus en plus rares (réplication totale). L'approche EcoTrans

est optimale quand le coût de transfert est relativement élevé. Cependant, EcoTrans ne modifie pas l'emplacement des fragments ce qui ne lui permet pas de converger vers une situation où les fragments fréquemment utilisés ensemble sont situés sur des sites différents.

Vus les avantages et les inconvénients de chaque approche, nous proposons une stratégie qui choisit dynamiquement la meilleure stratégie selon des requêtes. Pour cela, quand une requête arrive, nous comparons les coûts de ParaMax et Ecotrans et choisissons la stratégie dont le coût est moindre. De plus, afin de converger vers une situation où les fragments sont davantage répliqués, nous choisissons de façon périodique la stratégie ParaMax bien que sont coût soit plus élevé que EcoTrans. Le principe est de ralentir momentanément une requête afin d'accélérer le traitement des requêtes suivantes. La période est déterminé en fonction du ratio coût OL/ coût de transfert.

5.3.3.1. **Algorithme pour déterminer les sites participant à l'exécution d'une requête.**

Nous détaillons l'algorithme qui détermine quels sites participent à l'exécution d'une requête. Nous présentons le fonctionnement de l'algorithme pour chacune des deux stratégies Paramax et Ecotrans.

La stratégie ParaMax consiste à désigner autant de participants que de fragments. Si un site local possède déjà un fragment utile pour la requête, alors nous le choisissons autant que possible. Cependant, si un participant possède plusieurs fragments utiles, il traitera un seul fragment, les autres fragments seront transférés vers d'autres sites locaux. Par ailleurs, pour chaque fragment n'existant pas parmi les sites locaux, un site local est désigné pour prendre en charge le nouveau fragment. En définitive, l'algorithme garantit que le nombre de participants (i.e. le degré de parallélisme) est maximal.

Pour la stratégie EcoTrans, l'algorithme consiste à utiliser tous les sites locaux qui possèdent déjà au moins un fragment utile pour la requête. Un fragment pouvant être répliqué sur plusieurs sites, plusieurs associations $(fragment, participant)$ sont possibles. L'algorithme détermine les associations afin de répartir au mieux les fragments parmi les participants. Plus précisément, l'algorithme minimise le nombre maximum de fragments affectés sur un participant. Pour cela, nous considérons les fragments en commençant toujours par celui qui est le moins répliqué. Lorsque deux (ou plusieurs) sites contiennent un même fragment, nous associons en priorité le participant qui possède le moins de fragments (pour la requête) car l'autre site pourra être associé à d'autres fragments. On associe d'abord un premier fragment à chaque participant, puis un deuxième fragment et ainsi de suite jusqu'à ce que tous les fragments soient associés.

5.3.3.2. **Estimation du coût d'une requête**

Le coût d'une requête est donc égal au coût minimal entre les deux approches soit :

$C_R = Min(Max[Max(|F^j| * (T_{OL} + T_p)), |F^i| * T_{OL}], T_I + T_{OL} + T_p) + |F| * T_{OG}$

Où S_i est le site coordinateur du traitement et S_j un site participant aux traitements OL. Ceci permet de voir que notre approche offre de meilleurs temps de réponse que les deux autres approches et s'adapte à la meilleure indépendamment du contexte.

6. Validation expérimentale

Les principaux objectifs de la validation sont (i) de montrer la faisabilité de notre approche en implémentant les fonctions de transfert de fragments et de traitement de requêtes, et (ii) de montrer que notre approche offre de meilleures performances que les deux approches Paramax et Ecotrans, quel que soit le contexte,

6.1. Méthodes et outils

Nos expériences consistent à mesurer les temps de réponse pour notre stratégie de traitement de requêtes. Ces temps sont ensuite comparés avec les temps de réponse obtenus avec les stratégies Paramax et Ecotrans. Nous avons implémenté avec l'outil de simulation Simjava [21] le comportement des utilisateurs, des sites locaux, du portail GBIF ainsi que toutes les interactions entre les participants. Notre système est composé de 11 machines : une machine représente le GBIF et les dix autres correspondent aux sites locaux. Sur chaque site local, nous avons instancié notre service de requête. Nous avons implémenté l'algorithme déterminant le plan d'exécution. Nous avons aussi implémenté la coordination entre les participants pour le traitement réparti d'une requête. Chaque site local dispose d'un SGBD pour le traitement effectif d'une requête ou d'une sous-requête.

Nous avons généré un jeu de données (ensemble de fragments) qui est placé initialement sur le portail du GBIF. Autrement dit, les sites locaux ne contiennent initialement aucun fragment. Puis les fragments sont répliqués dynamiquement à la demande vers les sites locaux, en fonction des requêtes posées par les utilisateurs. Un utilisateur pose une séquence de requêtes tant que dure l'expérience. Il attend d'obtenir le résultat d'une requête avant de poser la suivante. Une requête accède dix fragments en moyenne.

Afin d'observer le bénéfice éventuel des stratégies Paramax et Ecotrans, nous mettons en place deux contextes différents. Dans le premier contexte, favorable à Paramax, le coût de transfert est est la moitié du coût de traitement local (OL) d'un fragment. Dans le deuxième contexte, favorable à Ecotrans, le coût de transfert est le double du coût de traitement local (OL) d'un fragment. Nous exécutons une expérience pour chaque stratégie (Ecotrans, Paramax et notre approche) dans chaque contexte.

6.2. Résultats obtenus

Les figures 5, 6 et 7 ci-dessous récapitulent les courbes d'évolution des temps de réponse pour une succession de requêtes d'utilisateur. Les figures indiquent aussi le temps de réponse moyen et le temps de réponse plafond pour 90% des requêtes de chaque expérience. Le premier constat est que les trois stratégies ont le même comportement initial, tant que les requêtes accèdent au GBIF (pendant environ 200.000 unités de temps). En effet, tant qu'un accès au GBIF est nécessaire, alors la requête est traitée de la même façon quelle que soit la stratégie . Par contre, lorsque les données sont toutes disponibles dans les sites locaux, les mécanismes de traitements et les temps de réponse diffèrent en fonction du contexte et de la stratégie utilisée pour traiter les requêtes.

Dans le premier contexte (i.e., coût de transfert relativement faible), notre stratégie décide toujours de paralléliser au maximum une requête, ce qui lui permet d'être aussi performante que la stratégie Paramax avec un temps de réponse moyen valant 275 unités de temps. Dans le deuxième contexte (i.e., coût de transfert relativement fort), notre stra-

tégie décide d'éviter les transferts et aboutit à des temps de réponse moyens (383) proches de ceux de la stratégie Ecotrans (381).

Lors de l'analyse approfondie des résultats de nos expériences, nous avons constaté que dans le deuxième contexte (i.e., le coût de transfert est le double du coût de traitement local) la stratégie Ecotrans ne s'avère pas toujours optimale. En effet, lorsque un participant est associé avec ou moins 4 fragments, le coût de traitement local (valant $4OL$) dépasse le coût d'un transfert (valant $2OL + OL = 3OL$). En conséquence, le plan d'exécution optimal ne dépend pas seulement du contexte mais aussi du placement des fragments au moment où la requête est posée. Cela renforce l'intérêt de notre solution qui se base sur le coût estimé d'une requête pour décider si un transfert est bénéfique.

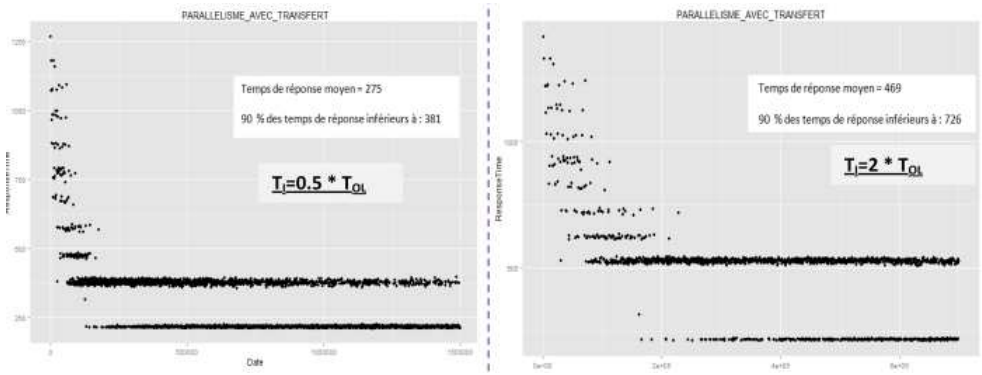

Figure 5. *Temps de réponse avec ParaMax*

En conclusion, notre solution s'adapte à tous les contextes qui se présentent afin d'offrir les meilleurs temps de réponse.

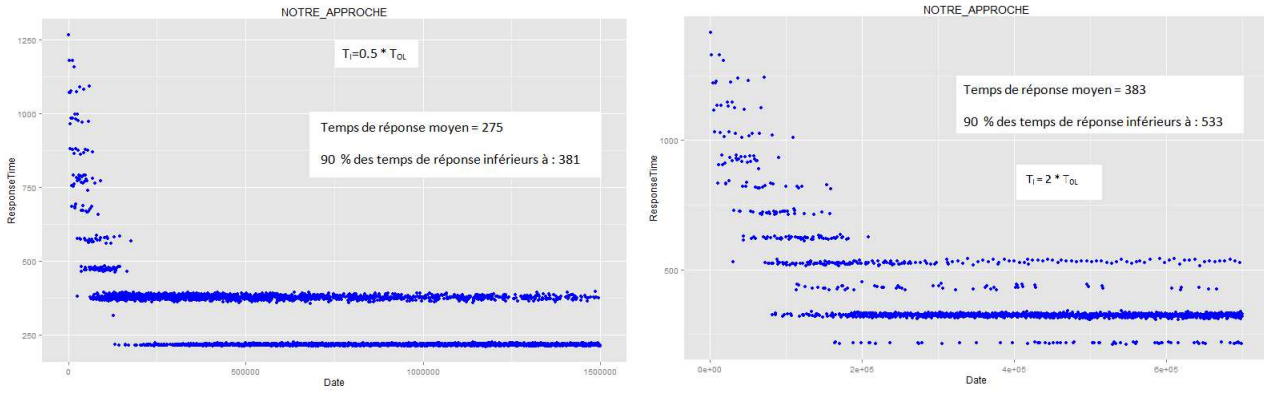

Figure 6. *Temps de réponse avec notre approche*

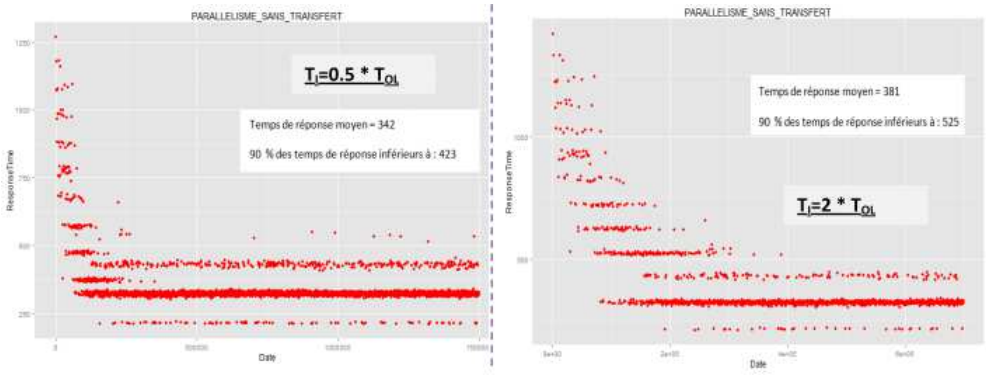

Figure 7. *Temps de réponse avec ecoTrans*

7. Conclusions et perspectives

Cet article présente une solution innovante et non intrusive, pour étendre les services d'accès offerts par une base de données en ligne (e.g., le portail GBIF) dans l'optique d'améliorer les performances liées au traitement des requêtes et de satisfaire les nouveaux besoins des utilisateurs. Nous avons proposé une décentralisation de l'accès aux données à travers plusieurs sites qui collaborent pour améliorer les performances des requêtes. Nous avons défini une architecture décentralisée pour traiter efficacement les requêtes des utilisateurs. Elle a l'avantage de fonctionner sans modifier le portail GBIF existant. Nous avons proposé une stratégie de répartition dynamique des données décentralisée ainsi que des algorithmes de traitement efficace des requêtes de biodiversité à travers l'architecture décentralisée. Nous avons instancié notre architecture et l'avons simulée sur un cluster de onze machines en nous appuyant sur des logiciels de gestion de donnés open source et récents. Ces expérimentations ont montré l'efficacité de notre approche d'adaptation au contexte pour le traitement des requêtes par rapport à d'autres stratégies qui pourraient être utilisées. Les résultats obtenus ont montré la faisabilité de notre solution et son efficacité pour des requêtes typiques de l'usage en biodiversité (calculs de co-occurrences d'espèces). Nous effectuons actuellement des expérimentations plus complètes, et mesurons le coût d'exécution des requêtes pour un grand nombre de sites avec des jeux de données réels du GBIF. Nous adaptons nos algorithmes aux spécificités d'un environnement hétérogènes de type cloud, telles que la forte fluctuation de la disponibilité des sites et de leur charge, l'hétérogénéité de leur capacité de calcul et des liens entre eux.

8. Bibliographie

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

[2] Azza Abouzied, Kamil Bajda-Pawlikowski, Jiewen Huang, Daniel J. Abadi, and Avi Silberschatz. HadoopDB in action : building real world applications. *ACM Intl Conf. on Management of data (SIGMOD)*, pages 1111–1114, 2010.

[3] Charu Aggarwal, Joel L. Wolf, and Philip S. Yu. Caching on the World Wide Web. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 11(1) :95–107, 1999.

[4] Divyakant Agrawal, Sudipto Das, and Amr El Abbadi. Big data and cloud computing : current state and future opportunities. In *Intl Conf. on Extending Database Technology (EDBT)*, pages 530–533, 2011.

[5] Peter M. G. Apers. Data Allocation in Distributed Database Systems. *ACM Transactions on Database Systems (TODS)*, 13(3) :263–304, 1988.

[6] Kamil Bajda-Pawlikowski, Daniel J. Abadi, Avi Silberschatz, and Erik Paulson. Efficient processing of data warehousing queries in a split execution environment. In *ACM Intl Conf. on Management of Data (SIGMOD)*, pages 1165–1176, 2011.

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

[8] D. Borthakur. The hadoop distributed file system : Architecture and design. *The Apache Software Foundation*, 2007.

[9] Matthias Brantner, Daniela Florescu, David A. Graf, Donald Kossmann, and Tim Kraska. Building a database on s3. In *ACM Intl Conf. on Management of Data (SIGMOD)*, pages 251–264, 2008.

[10] Fay Chang et al. Bigtable : A Distributed Storage System for Structured Data (Best Paper Award). In *USENIX Symp. on Operating System Design and Implementation (OSDI)*, pages 205–218, 2006.

[11] Deniz Cokuslu, Abdelkader Hameurlain, Kayhan Erciyes, and Franck Morvan. Resource allocation algorithm for a relational join operator in grid systems. In *Intl Database Engineering & Applications Symposium (IDEAS)*, pages 139–145, 2012.

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

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

[14] GBIF France and Canadian BIF. Gbif france, canadian bif, 2013.

[15] GBIF. Gbif data use case.

[16] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google file system. In *ACM Symp. on Operating Systems Principles (SOSP)*, pages 29–43, 2003.

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

[18] Alon Y. Halevy. Answering queries using views : A survey. *VLDB Journal*, 10(4) :270–294, 2001.

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

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

[21] Fred Howell and Ross Mcnab. simjava : A discrete event simulation library for java. In *Intl Conf. on Web-Based Modeling and Simulation*, pages 51–56, 1998.

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

[23] D. Kossmann et al. The state of the art in distributed query processing. *ACM Computing Surveys*, pages 422–469, 2000.

[24] 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.

[25] Mengmeng Liu. Efficient optimization and processing for distributed monitoring and control applications. In *ACM SIGMOD/PODS PhD. Symposium*, pages 69–74, 2012.

[26] Thanasis Loukopoulos and Ishfaq Ahmad. Static and adaptive data replication algorithms for fast information access in large distributed systems. In *Intl Conf. on Distributed Computing Systems (ICDCS)*, pages 385–392, 2000.

[27] MosquitoMap. www.mosquitomap.org.

[28] Map of Life. www.mappinglife.org.

[29] S. Poldlipnig and L. B Osz Ormenyi. A survey of web cache replacement strategies. *ACM Computing Surveys*, 35(4) :374–398, 2003.

[30] L. Rupprecht. Exploiting in-network processing for big data management. In *ACM SIGMOD/PODS PhD. Symposium*, pages 1–6, 2013.

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

[32] GBIF Secretary. The gbif data portal : A practical « hands-on » accessible au http ://data.gbif.org.

[33] GBIF Secretary. Gbif data portal, gbif web site, 2013.

[34] A. Thusoo et al. Hive - A Petabyte Scale Data Warehouse Using Hadoop. In *Intl Conf. on Data Engineering (ICDE)*, pages 996–1005, 2010.

[35] Reynold S. Xin, Josh Rosen, Matei Zaharia, Michael J. Franklin, Scott Shenker, and Ion Stoica. Shark : SQL and rich analytics at scale. In *ACM Intl Conf. on Management of Data (SIGMOD)*, pages 13–24, 2013.

[36] Jing Zhao, Xiangmei Hu, and Xiaofeng Meng. ESQP : an efficient SQL query processing for cloud data management. In *CIKM Intl Workshop on Cloud Data Management (CloudDB)*, pages 1–8, 2010.