Relations complexes au sein du système Hive

Matthieu Pilven[*,**] Stefanie Scherzinger[*]
Laurent d'Orazio[**]

[*]OTH Regensburg, Regensburg, Germany
stefanie.scherzinger@oth-regensburg.de,
[**]Univ Rennes, CNRS, IRISA, France
laurent.dorazio@irisa.fr

Résumé. Dans cet article, nous abordons le problème de la modélisation de données pour les entrepôts au sein du système Hive. Ce moteur SQL sur Hadoop bien connu permet des relations avec des valeurs complexes, avec différents types d'attribut non nécessairement atomiques. A l'aide d'une étude empirique, nous analysons des dossiers de sources libres qui contiennent des déclaration de schémas Hive. Nous examinons dans quelles limites les utilisateurs créent des relations à valeur complexe et par conséquent leurs requêtes sur les valeurs complexes. Comprendre comment les utilisateurs conçoivent leurs modèles de données Hive, en particulier quelles fonctionnalités ils utilisent, devrait nous aider à prendre de bonnes décisions de conception pour la création d'un banc d'essais réaliste pour les moteurs SQL sur Hadoop, et choisir quels opérateurs de requêtes utiliser pour une bonne optimisation.

1 Introduction

A l'origine un entrepôt de données interne à Facebook Thusoo et al. (2009), le moteur SQL sur Hadoop Hive est désormais un projet officiel Apache. Après seulement une décennie de développement, Apache Hive figure parmi les 10 premiers systèmes de bases de données relationnelles sur le site Web DB-Engines Ranking [1], aux côtés des bases de données de DB2 ou Oracle. Pourtant, il s'agit de produits commerciaux dont l'historique de développement est jusqu'à quatre fois plus long. Une partie de ce succès est due au fait que travailler avec Hive est très familier :

— Les utilisateurs de Hive interagissent facilement avec un modèle de données relationnel, mais les données brutes sont généralement stockées dans le système de fichiers distribués de Hadoop (HDFS).
— Bien que le langage de requête HiveQL ressemble étroitement au dialecte SQL de MySQL (White, 2009), les requêtes sont exécutées sous forme de tâches MapReduce évolutives.

De plus, Les relations Hive n'ont pas forcément besoin d'être en première forme normale, où toutes les valeurs d'attributs sont atomiques. Au contraire, le modèle de données permet

1. https://db-engines.com/de/ranking/relational+dbms, de Juin 2019.

```
1  CREATE TABLE employees (
2    name STRING,
3    salary FLOAT,
4    subordinates ARRAY<STRING>,
5    deductions MAP<STRING, FLOAT>,
6    address STRUCT<street:STRING, CITY:STRING, state:STRING, zip:INT>
7  );
8   %
9  SELECT name, subordinates[0] FROM employees;
10 SELECT explode(subordinates) AS sub FROM employees;
11 %
12 SELECT name, sub FROM employees
13 LATERAL VIEW explode(subordinates) subView AS sub;
```

FIG. 1 – *Une déclaration de table Hive et des requêtes, tirées de Capriolo et al. (2012).*

(pour une forme restreinte) des relations de valeurs complexes : un constructeur de n-uplets `struct` déclare des n-uplets. Les types complexes `map` et `array` déclarent respectivement des tableaux associatifs ou indexés, des concepts familiers aux langages de programmation. Ces constructeurs peuvent être appliqués de manière récursive, à un niveau d'imbrication arbitraire, afin qu'un architecte de données puisse déclarer un tableau de `map` ou de `structs`. Cela permet l'ingestion de données qui arrivent sous forme imbriquée (par exemple, au format JSON). En outre, la dénormalisation accélère le traitement des requêtes dans les paramètres de l'entrepôt de données, où les données sont rarement mises à jour (voire pas du tout). Les relations de valeurs complexes sont aussi supportées dans les systèmes connexes, c.f. Impala Kornacker et al. (2015) et Presto [2].

Par exemple, la déclaration de table de la Figure 1 assimile les employés avec leur nom, leur salaire et leurs subordonnés imbriqués. Elle prend également en compte les déductions fiscales spécifiques aux employés, ainsi que leur adresse. Cette exemple vient de Capriolo et al. (2012). La syntaxe pour accéder à un champ dans un tableau est simple, comme indiqué à la ligne 9. HiveQL propose des fonctions de génération de tables, de ce fait, les requête à la ligne 10 créent un n-uplet pour chaque élément de du tableau `subordinates`. Pour lister les paires d'un responsable et d'un subordonné, nous devons déclarer un `LATERAL VIEW`, comme indiqué aux lignes 12 et 13. □

Contributions. Nous explorons les relations de valeurs complexes dans Hive en analysant des dossiers de sources libres disponibles sur GitHub. Il existe une tradition d'étude empirique sur les schémas de bases de données dans les projets de sources libres, généralement dans le contexte d'évolution des schémas, par exemple Curino et al. (2008); Qiu et al. (2013). Cependant, à notre connaissance, il n'existe aucune étude sur la diffusion de relations de valeurs complexes dans les moteurs SQL basés sur Hadoop. En particulier,

— Nous formalisons les relations de valeurs complexes dans Hive et montrons les connexions à la théorie de V-relations Abiteboul et al. (1995) qui date des années 80.

— Nous identifions 133 projets GitHub uniques et pertinents avec un total de plus de 900 déclarations de tables. Nous identifions ensuite les relations à valeur complexes dans les

2. `https://prestosql.io/`

schémas Hive, ainsi que les occurrences des opérateurs correspondants dans les requêtes HiveQL.

— Par analogie, nous explorons les bancs d'essais de bases de données existants, en particulier si les tests conçus pour Hive correspondent à ce que nous avons trouvés dans notre analyse. Cela pourrait donner des impulsions à la conception de futurs bancs d'essais pour les données massives.

Il est à noter que ce travail constitue une version préliminaire d'un article présenté à MOBID@ER2019.

Organisation. Dans la Section 2, nous fournissons les préliminaires. Dans la Section 3, nous établissons la méthodologie de notre étude, y compris nos questions de recherche. Dans la Section 4, nous présentons nos résultats d'étude détaillés, qui sont discutés dans la Section 5. Nous listons les menaces à la validité de notre étude dans la Section 6. La Section 7 donne un aperçu des travaux connexes. Nous concluons ensuite avec la Section 8.

2 Préliminaires

Nous récapitulons d'abord les définitions de base des relations de valeur complexes, puis formalisons des types complexes dans Hive. Enfin, nous montrons comment les types Hive peuvent être vus comme des spécialisations de relations de valeur complexes.

Relations à valeur complexes Abiteboul et al. (1995) Dans les relations à valeur complexes, les attributs ont besoin d'être atomiques. Intuitivement, les structures de données pour les relations à valeur complexes utilisent deux constructeurs, qui peuvent être utilisés récursivement. (1) un *constructeur de n-uplets* pour faire des n-uplets (2) Un *constructeur d'ensembles* faire des ensembles de n-uplets, et donc des relations. Sous la notion de schéma, il y a la notion de types complexes (ou *sorts*). Pour spécifier une instance, nous spécifions des ensembles de valeurs complexes correspondant à ces types. Comme dans le modèle relationnel, nous supposons un ensemble de constantes dom. La syntaxe abstraite pour les types complexes est illustrée ci-dessous :

$$\tau \;=\; \mathbf{dom} \mid \langle B_1 : \tau, \ldots, B_k : \tau \rangle \mid \{\tau\},$$

Où $k \geq 0$, et $B_1, \ldots, B_k$ sont des noms d'attributs distincts. Intuitivement, un élément du **dom** est une constante, un élément de $\langle B_1 : \tau, \ldots, B_k : \tau \rangle$ est un k-uplet avec un élément de type τ_i en entrée B_i pour chaque i.

Nous nous référons à Abiteboul et al. (1995) pour les définitions formelles de $[\![\tau]\!]$, l'ensemble de valeurs de type τ, et simplement introduit à titre d'exemples. Nous définissons une *relation à valeur complexes de type* τ comme étant un ensemble de valeurs finies de type τ.

Exemple 1 *Pour la Figure 1 (jusqu'à la ligne 3), le type (abstraction de* **string** *et* **float***) est* $\{\langle \text{name} : \mathbf{dom},\ \text{salary} : \mathbf{dom} \rangle\}$. *une valeur de ce type est déclarée ainsi :* $\{\langle \text{name} : \texttt{"John Doe"}, \text{salary} : 100000.0 \rangle,\ \langle \text{name} : \texttt{"Mary Smith"}, \text{salary} : 80000.0 \rangle\}$. □

Exemple 2 *Ci-dessous, nous considérons un type complexe pour déclarer une relation à valeurs complexes non plates (equation 1), et une valeur pour ce type (equation 2) :*

$$\{\langle \text{name} : \mathbf{dom},\ \text{salary} : \mathbf{dom},\ \mathbf{subordinates} : \{\langle \text{key} : \mathbf{dom}, \text{value} : \mathbf{dom}\rangle\}\rangle\}. \tag{1}$$

$$\begin{aligned}&\{\langle \text{name} : \texttt{"John Doe"}, \text{salary} : 80000.0, \\ &\quad \text{subordinates} : \{\langle \text{key} : 0, \text{value} : \texttt{"Mary Smith"}\rangle,\ \langle \text{key} : 1, \text{value} : \texttt{"Tod Jones"}\rangle\}\rangle, \\ &\langle \text{name} : \texttt{"Mary Smith"}, \text{salary} : 80000.0, \text{subordinates} : \{\}\rangle, \\ &\langle \text{name} : \texttt{"Todd Jones"},\ \text{salary} : 70000.0, \text{subordinates} : \{\}\rangle\}\end{aligned} \tag{2}$$

□

La Figure 2 montre une visualisation du type complexe à partir de l'exemple ci-dessus sous forme d'arbre fini. Le constructeur du n-uplet est désigné par un nœud nommé $\times$ et le constructeur de l'ensemble par un nœud intitulé $*$. Les bords sortants des nœuds du n-uplet sont étiquetés, tandis que les nœuds d'ensemble ont un seul enfant. En se basant sur cette visualisation d'arbre, il est très intuitif de définir la *hauteur de l'ensemble* Abiteboul et al. (1995) d'un type complexe comme le nombre maximal de constructeurs d'ensemble dans n'importe quelle branche. Dans l'arbre montré, la hauteur de l'ensemble est 2

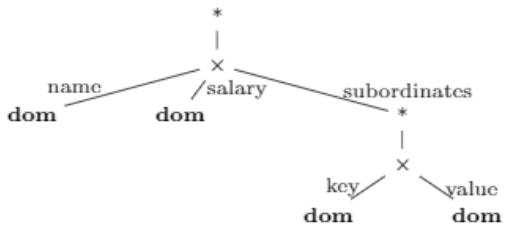

FIG. 2 – *Type à valeurs complexes de l'exemple 3, représenté sous forme d'arbre avec une hauteur de l'ensemble de 2.*

Les types dans Hive. Le modèle de données Hive propose un deuxième constructeur de type via `struct`. Bien qu'il ne permette pas d'imbriquer avec un constructeur d'ensembles approprié, nous pouvons imbriquer des `maps` et des `arrays`. Nous déclarons la syntaxe abstraite pour les *type dans Hive* τ_H telle que :

$$\tau_H = \{\langle B_1 : \tau, \ldots, B_k : \tau\rangle\} \tag{3}$$

$$\tau = \mathbf{dom} \mid \texttt{map} <\tau, \tau> \mid \texttt{array} <\tau> \mid \texttt{struct} <B_1 : \tau, \ldots, B_k : \tau>. \tag{4}$$

Ainsi, le type du niveau au dessus est toujours un ensemble de n-uplets. En dessous, les `maps`, les `arrays` et les `structs` peuvent être imbriqués de manière arbitraire. Etant donné une relation plate de Hive, nous disons que le *niveau de profondeur de Hive* est 1. Pour toute déclaration (récursive) d'un `map`, d'un `array` ou d'un `struct`, le niveau d'imbrication de Hive augmente de un. L'ensemble de valeurs d'un type Hive τ_H est noté par $[\![\tau_H]\!]$ et déclaré

après. Ci-dessous, nous assimilons struct avec le constructeur de n-uplet. Les `arrays` et les `maps` sont codés sous forme d'ensembles de paires clé-valeur :

$$\begin{aligned}
[\![\texttt{struct} <B_1 : \tau_1, \ldots, B_k : \tau_k>]\!] &= [\![\langle B_1 : \tau_1, \ldots, B_k : \tau_k \rangle]\!] \\
[\![\texttt{map} <\tau_k, \tau_v>]\!] &= \{\{\langle \text{key} : k_1, \text{value} : v_1\rangle, \ldots, \langle \text{key} : k_j, \text{value} : v_j\rangle\} \\
&\quad \mid j \geq 0, k_i \in [\![\tau_k]\!], v_i \in [\![\tau_v]\!], i \in [1, j]\} \\
[\![\texttt{array} <\tau>]\!] &= \{\{\langle \text{key} : 1, \text{value} : v_1\rangle, \ldots, \langle \text{key} : j, \text{value} : v_j\rangle\} \\
&\quad \mid j \geq 0, v_i \in [\![\tau]\!], i \in [1, j]\}
\end{aligned}$$

Ensuite, une *relation Hive de type* τ_H est une valeur de type τ_H.

Les relations Hive par rapport aux relations de valeur complexes. Le modèle de données Hive ne permet pas l'imbrication avec un constructeur d'ensemble donné, une limite a aussi été observée dans une enquête qualitative sur le moteur SQL sur Hadoop Sauer et Härder (2013). Néanmoins, nous pouvons établir un lien entre les relations de Hive et la relation de valeur complexe : Les définitions de valeurs d'un type Hive peuvent être généralisées comme valeur complexe. Par exemple, il serait possible de généraliser un `array` comme un ensemble de n-uplets clé-valeur :

$$[\![\texttt{array} <\tau>]\!] \subset [\![\{\langle \text{key} : \mathbf{dom}, \text{value} : \tau\rangle]\!].$$

Il convient de noter que le type généralisé autorise les index de tableau non consécutifs et même répétitifs, il définit donc vraiment un sur-ensemble de valeurs. Compte tenu de cette généralisation, nous pouvons relier les relations de Hive à un modèle de données exploré dans des recherches antérieures : nous pouvons supposer en toute sécurité que toutes les imbrications récursives de `structs` avec des `structs` sont normalisées à un seul `struct`. Cela peut être fait sans perte d'information. Ensuite, la généralisation ci-dessus des types complexes de Hive en types à valeur complexes produit des relations VersoAbiteboul et Bidoit (1984). Ces structures de données ont la propriété bénéfique que les informations contenues peuvent être représentées de manière équivalente en utilisant plusieurs relations plates. Cela impose une contrainte polynomiale à la cardinalité d'un ensemble dans une relation Verso, ce qui est une propriété intéressante pour l'évaluation pratique des constructeurs de n-uplets, tels que `EXPLODE`.

3 Méthodologie

Nous exposons ensuite nos questions de recherche et décrivons notre méthodologie. Nous renvoyons à la Section 6 pour une analyse des menaces à la validité de nos résultats.

3.1 Description du contexte

Nous avons utilisé Google BigQuery [3] pour identifier les projets de sources libres pertinents sur GitHub, datés du 17 juin 2019. Ce service permet d'interroger la collection de données ouverte GitHub, principalement des dossiers non dupliqués (forks) avec une licence de source libre. Nous considérons un dossier pertinent s'il possède au moins un fichier finissant

3. `https://cloud.google.com/bigquery/`

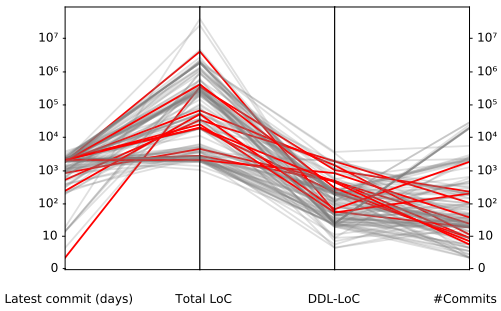

FIG. 3 – *Vue d'ensemble des caractéristiques des 133 dossiers analysés.*

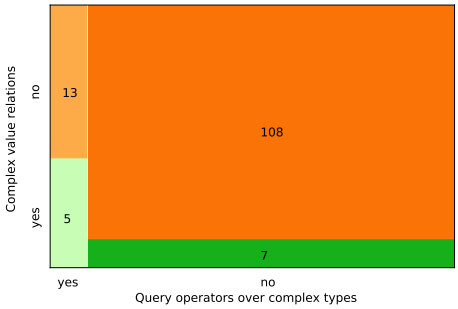

FIG. 4 – *Projets avec des relations de valeur complexes vs.\requêtes sur des types complexes.*

par `.hql`, qui désigne généralement un fichier avec des instructions HiveQL. Cela a révélé 414 dossiers. Nous avons éliminé tous les projets enregistrés comme des forks d'autres projets, ce qui en laisse 158. Nous avons également éliminé 25 dossiers sans déclaration de table dans les fichiers `hql`. Au total, il nous reste 133 dossiers pour une analyse détaillée. La Figure 3 permet de visualiser les caractéristiques clés des dossiers analysés (à ce stade, les couleurs sont ignorées) : nous rapportons le nombre total de lignes de code (LoC), les jours passés depuis le dernier commit et les lignes de code pour les déclarations de table dans les fichiers hql. Nous appelons cela les *DDL-LoC* (c.f. Section 3.3), et le nombre de commits. Les lignes de code des déclarations de schéma sont généralement considérées comme une *métrique proxy* pour la complexité du schéma. Jain et al. (2016). Évidemment, les caractéristiques varient énormément.

3.2 Questions de Recherche

Nous formulons maintenant nos questions de recherche, que nous abordons ensuite à la section 4 :

RQ1 : Quelle est la fréquence des relations de valeurs complexes dans les schémas Hive ?
RQ2 : Quel est l'usage des opérateurs de requête sur des valeurs complexes ?
RQ3 : Les bancs d'essais de bases de données considèrent-ils des relations avec valeurs complexes ?

3.3 Processus d'analyse

Analyser les schémas de Hive. Nous avons écrit un parser basé sur Python pour le traitement des fichiers `hql`. Dans un des fichiers `hql` le point virgule étant absent comme séparateur, nous avons corrigé cela manuellement pour inclure ce fichier dans notre analyse. Nous extrayons ensuite toutes les déclarations de `CREATE TABLE` et structurons avec une fonction `pretty-print` de telle sorte qu'un attribut est déclaré à chaque ligne, dans un but de normalisation. Deux lignes sont toujours requises pour ouvrir et fermer la déclaration de

table respectivement, comme montré dans les lignes 1–7 de la Figure 1. Il convient de noter que nous ignorons toutes les déclarations de la forme `CREATE TABLE LIKE <T>` et `CREATE TABLE AS <Q>`, dont nous trouvons respectivement 17 et 117 occurrences Ainsi, nous ne tenons pas compte de 11% des déclarations de `CREATE TABLE`. Nous justifions cette approche comme suit : avec le premier constructeur, nous créons des copies de tableaux que nous avons déjà analysés. Il y a donc peu de valeur ajoutée. Le second constructeur est couramment utilisé pour stocker des résultats intermédiaires dans des chaînes de traitement, comme on le ferait une vue matérialisée. Nous soutenons que ce ne sont pas les tables de base contenant les données d'origine et nous choisissons donc de les ignorer dans notre analyse. Suivant la même logique, nous n'analysons pas les déclarations `CREATE VIEW`, où des valeurs complexes pourraient être introduites en imbriquant des valeurs provenant de relations de base plates.

Analyser les requêtes HiveQL. Le langage de requêtes de Hive propose plusieurs constructeurs pour traiter des types complexes, [4] tels que

— des fonctions de collection, e.g. `array_contains(A,v)` détermine si le tableau `A` contient une valeur donnée `v`,
— des constructeurs de types complexes, e.g. (pour créer des `maps`), et des
— fonctions intégrées de génération de table. [5]

Nous appliquons l'opérateur `grep` sur les instructions `SELECT` avec ces constructions syntaxiques dans les fichiers `hql`. Nous excluons les méthodes d'accès de champs uniques, telles que `A[i]` pour accéder au $i$ème élément d'un array `A`, ou `S.k` pour accéder à l'élément `k` de la structure `S`. Nous ignorons en outre les fonctions intégrées définies par l'utilisateur pour les requêtes de style XPath.

4 Résultats détaillés

4.1 RQ : Quelle est la fréquence des relations à valeurs complexes dans les schémas Hive ?

Dans ce qui suit, lorsque nous mentionnons des relations à valeurs complexes, nous supposons des relations *non-plates* et nous nous référons explicitement aux relations plates.

Seul 9% des dossiers analysés contiennent des relations de valeurs imbriquées. Le Tableau 1 fournit des détails : à côté du dossier, nous indiquons combien parmi les relations analysées sont des relations à valeurs complexes non-plates (colonne#CVR). Par exemple, pour le premier dossier, 3 sur 16 relations contiennent des types complexes. Ensuite, nous déclarons le niveau d'imbrication Hive maximale (HNL) et la hauteur définie (SH). Nous spécifions ensuite le nombre de contributeurs (#Cont) pour chaque dossier GitHub.

Bien que Hive ne limite pas le niveau d'imbrication, le maximum de HNL observé n'est que de 3, et donc assez peu profond.

La hauteur définie est 2 dans tous les dossiers sauf un, il n'y a donc pas d'imbrication répétitive de tableaux.

4. https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF
5. https://cwiki.apache.org/confluence/display/Hive/LanguageManual+XPathUDF

TAB. 1 – *The subset of repositories with (non-flat) complex value relations.*

	Repository	#CVR	HNL	SH	#Cont
1	sixeyed/hive-succinctly	3 out of 16	3	2	1
2	jbrambleDC/predict_restaurant_success	2 out of 3	3	2	1
3	flaminem/flamy	21 out of 51	2	2	1
4	yhemanth/hive-samples	1 out of 22	2	2	1
5	Benjguin/UnlockLuxury	1 out of 15	2	2	2
6	DXFrance/data-hackathon	1 out of 15	2	2	1
7	mellowonpsx/ESCA	2 out of 7	2	1	1
8	Sicmatr1x/Sicmatr1x.github.io	1 out of 7	2	2	1
9	PolymathicCoder/Avempace	4 out of 5	2	2	1
10	airbnb/aerosolve	1 out of 4	2	2	21
11	gliptak/hadoop-course	1 out of 4	2	2	1
12	EXEM-OSS/Flamingo2	1 out of 3	2	2	1

Une inspection manuelle révèle que seuls quelques dossiers contiennent des projets de science des données : le dossier 2 prédit le succès des restaurants en fonction de différentes fonctionnalités. Il prend les attributs du restaurant, tels que l'ambiance (romantique, intime, chic, ...) et les options de stationnement (garage, rue, ...), en utilisant des structures imbriquées pour coder un vecteur de Booléens. Le dossier 10 est une librairie d'apprentissage populaire où les relations de valeur complexes contiennent des données d'apprentissage. Les dossiers restants semblent être expérimentaux, ce qui est aussi suggéré par des noms de dossiers tels que `hadoop-course` et `hive-samples`, certains suivent actuellement un tutoriel Hive avec exactement la même déclaration de table illustrée par la Figure 1. Ainsi, la plupart de ces projets semblent être pour le développement expérimental et personnel : seul `aerosolve` se démarque avec 21 contributeurs.

Comme nous le verrons, cet effet est prévisible lors de l'analyse des référentiels GitHub Kalliamvakou et al. (2014).

Une inspection manuelle du dossier 7 révèle qu'il y a une relation à valeurs complexes avec un HNL 3 dans un fichier `sql` (et non `hql`). Nous nous référons à la Section 6 pour une discussion de cet effet.

La Figure 3 montre les 12 dossiers dans le contexte des dossiers restants (il est recommandé de regarder l'image en couleur ou sur un écran), la ligne rouge montre les dossiers du Tableau 1. Deux points sont communs aux dossiers du tableau. (1) Le dernier commit remonte à six mois ou plus. Ainsi, tous les projets en cours de développement actif n'ont que des relations plates. (2) En outre, tous les référentiels du tableau 1 ont des déclarations de schéma couvrant au moins 57 lignes de code dans leur déclaration de schéma et sont donc supérieures à 40% des référentiels étudiés.

Résultats Nous trouvons étonnamment peu de preuves de relations à valeurs complexes. En fait, dans la majorité des cas, les développeurs expérimentent simplement des types complexes. De plus, même si Hive ne limite pas le niveau d'imbrication, le maximum observé est 3. Bien entendu, des types complexes peuvent également être introduits dans les vues, ce qui est un effet que nous pouvons observer lorsque nous étudions la deuxième question de recherche.

4.2 RQ : Quel est l'usage des opérateurs de requête sur des valeurs complexes ?

Dans tous les référentiels, nous avons analysé 2 764 requêtes HiveQL dans des fichiers hql. Nous analysons ces requêtes pour les opérateurs impliquant des types complexes (voir Section 3.3 pour les détails). Dans la Figure 4 nous pouvons voir un diagramme en mosaïque, où nous comparons l'usage de Relations à Valeur Complexes (CVR) (yes/no) Avec l'usage d'opérateurs requêtes de Valeurs Complexes (CV) (yes/no) dans les requêtes Hive de tous les dossiers. Comme attendu, la plus grande zone avec 110 dossiers représente ceux qui n'ont ni CVR ni opérateur de requêtes de CV, ceci étant lié à la rare utilisation de l'imbrication. Seulement 3 projets utilisent les deux, avec une majorité d'opérateurs `explode` et `lateral_view`. 5 projets utilisent l'imbrication mais n'ont aucune requête ni opérateur spécifique pour l'exploiter. Enfin, nous avons 6 projets avec des opérateurs mais aucune table avec des CVR. 3 d'entre eux ont un `explode` utilisé sur une vue au lieu d'une table, que nous n'analysons donc pas, comme indiqué dans la Section 3.3. Il y a aussi des opérateurs `lateral_view` combinés avec un `json_tuple`, et des opérateurs `size`.

Ainsi, l'opérateur le plus utilisé est `lateral\_view`, ce qui est naturel car il peut être combiné avec l'opérateur `explode` et l'opérateur `json_tuple`. Ensuite, c'est logiquement `explode`, celui étant l'opérateur le plus connu pour diviser une fonction de type imbriquée. `json\_tuple` est comme `explode`, mais une version plus efficace que `get_json_object`, fait spécifiquement pour extraire des n-uplets json. Enfin, l'opérateur `size` est le moins utilisé dans cette liste.

4.3 RQ : Les bancs d'essais de bases de données considèrent-ils des relations de valeur complexes ?

L'utilisation de bancs d'essais a toujours était très importante dans la recherche en gestion de données Cooper et al. (2010),DeWitt (1993),Huang et al. (2010),Pavlo et al. (2009),Poess et al. (2017).

Le Tableau 2 présente un aperçu des principaux bancs d'essais en général, et en masses de données en particulier. Les Micro-benchmarks, pour évaluer les opérations de systèmes de fichiers distribués sur des clusters modernesShankar et al. (2014) sont en dehors du domaine ciblé par ce travail.

Le Tableau 2 distingue les bancs d'essais axés sur le test de fonctions spécifiques (F) et les bancs d'essais qui spécifient un scénario d'application complet (A). Des exemples pour un banc d'essais de fonctions sont le tri (`terasort`) ou évaluation d'opérateurs SQL spécifiques (`select`, `project`, `join`, `order-By`, etc.). Par exemple, GridMix [6] et PUMA [7] appartiennent à cette catégorie. Les bancs d'essais axés sur les Applications mesurent généralement la performance des systèmes (matériels et logiciels), avec des jeux de données et des charges de travail donnés. Parmi eux, il existe des bancs d'essais utilisés pour étudier les systèmes de gestion de base de données traditionnels (SGBD) comme le banc d'essais Wisconsin DeWitt (1993), ou beaucoup les bancs d'essais de la famille TPC [8], tel que TPC-C, TPC-E, TPC-DI.

6. https ://hadoop.apache.org/docs/r1.2.1/gridmix.html
7. https ://github.com/yncxcw/PumaBenchmark4NewAPI
8. http ://www.tpc.org/

TAB. 2 – *An overview over well-known database benchmarks.*

Benchmark	CVR-like data model	Queries over complex types	Targeted at Hive
GridMix	◯	◯	◯
PUMA	◯	◯	◯
Wisconsin	◯	◯	◯
TPC-C, TPC-E, TPC-DI	◯	◯	◯
TPC-H, TPC-DS, Hive-600	◯	◯	✓
HiBench, SmartBench	◯	◯	✓
Pavlo et al	◯	◯	✓
Hive-testbench	◯	◯	✓
YCSB	◯	◯	◯
Yahoo ! Streaming Benchmark	✓	◯	◯
PigMix	✓	✓	◯
BigBench	✓	✓	✓
TPC-xbb	✓	✓	✓
UniBench	✓	✓	✓

Des bancs d'essais ont été proposés pour le traitement de masses de données en particulier. Certains se limitent aux données structurées (relationnelles) : Yahoo ! Benchmark Serving Cloud (YCSB) Cooper et al. (2010), TPC-H, TPC-DS Poess et al. (2017), Hive-testbench [9], Hive-600 [10] et Pavlo et al. Pavlo et al. (2009).

HiBenchHuang et al. (2010) et SmartBench [11] sont des suites de bancs d'essais axées à la fois sur les fonctions et les applications, pour étudier différents types de charges de travail (micro, apprentissage automatique, etc.). Ils considèrent une charge de travail de requêtes SQL, mais avec des limites en termes d'expressivité.

Les requêtes spécifiques de Hive sur des données semi-structurées sont prises en compte dans PigMix, PC-xbb et UniBench. PigMix consiste en un ensemble de requêtes permettant de tester les performances de Pig. Il prend en compte les fonctions intégrées de Pig, permettant d'accéder à des données imbriquées, par exemple via une fonction `explode`. TPC-xbb vise les systèmes basés sur Hadoop (et Hive entre autres), et couvre plusieurs modèles de données.

UniBenchZhang et al. (2018) est un banc d'essais pour les systèmes de gestion de bases de données multimodèles, prenant ainsi en compte les tables traditionnelles mais également les documents ou les graphes. Il étudie en particulier les données imbriquées. Comme indiqué par le Tableau 2, ces trois derniers bancs d'essais sont les seuls à interroger des données imbriquées.

Resultats. Il ressort clairement de cet examen approfondi des bancs d'essais que très peu d'entre eux contiennent des données imbriquées et des requêtes sur des données imbriquées. Cela est certainement dû aux efforts des concepteurs de bancs d'essais, qui, afin de fournir une plate-forme équitable de comparaison des systèmes, choisissent le plus petit dénominateur commun pour la fonctionnalité du système. Ainsi, les bancs d'essais communs ne reflètent généralement pas des relations de valeur complexes et des requêtes sur celles-ci.

9. https ://github.com/hortonworks/hive-testbench
10. https ://issues.apache.org/jira/browse/HIVE-600
11. https ://github.com/bomeng/smartbench

5 Discussion

Dans les dossiers analysés, avec plus de 90%, la majorité des schémas est en réalité en première forme normale. Bien que nous ayons effectué une analyse sur les dossiers de sources libres uniquement, cela correspond à nos impressions suite à des discussions avec les utilisateurs de Hive. Il existe plusieurs hypothèses sur les raisons : **(1)** une des raisons peut être que lorsque des données sont ingérées dans Hive à partir d'un entrepôt de données relationnel, les données sont "plates" pour commencer ; **(2)** une autre hypothèse est que plusieurs outils peuvent accéder aux mêmes données dans HDFS, mais ils ne sont pas tous capables de gérer des types complexes. Ainsi, les architectes de données peuvent être moins enclins à introduire des relations à valeurs complexes. **(3)** Les requêtes sur des relations à valeurs complexes ont tendance à devenir plus complexes que celles sur des relations plates. Bien que l'instruction `explode` soit simple, nous avons constaté que travailler avec des `lateral views` est moins intuitif pour les novices, du moins selon nos observations répétées auprès d'étudiants. Avec des niveaux d'imbrication plus profonds, les auteurs de requêtes ont également besoin de modèles mentaux complexes pour l'écriture de requêtes. En fait, le premier dossier du tableau 1 inclut une vue "plate" sur une relation à valeurs complexes, probablement pour faciliter la formulation de la requête.

Néanmoins, nous avons trouvé des relations avec valeurs imbriquées dans la pratique. De plus, comme évoqué précédemment, celles-ci se retrouvent dans certains scénarios de bancs d'essais pour moteurs SQL basés sur Hadoop, qui considèrent notamment Hive.

6 Menaces à la validité

Validité de construction. **(1)** En identifiant les dossiers pertinents sur GitHub, nous supposons que les déclarations `CREATE TABLE` sont toujours contenues dans des fichiers `hql`. Cependant, il ne s'agit que d'une convention. Les fichiers peuvent être nommés différemment, et les déclarations `CREATE TABLE` pourrait être intégrées dans le code de l'application. Bien que cela soit certainement une limite de notre méthodologie, ignorer les instructions SQL dans le code d'application est une pratique courante dans pratiquement toutes les études empiriques antérieures sur les schémas relationnels, e.g. Qiu et al. (2013); Curino et al. (2008). Ainsi, il existe des schémas de Hive que nous ne considérons pas et il est souhaitable d'élargir notre recherche dans une analyse future à plus grande échelle. En même temps, si un dossier basé sur Impala contient des fichiers hql, nous inclurons faussement cela dans notre analyse. Cependant, nous sommes assez confiants dans nos résultats, car nous avons soigneusement inspecté les dossier du Tableau 1 afin de détecter les signes indiquant qu'il ne s'agissait peut-être pas de projets Hive.

(2) Nous n'analysons pas actuellement les instructions ALTER TABLE. Ainsi, si une instruction CREATE TABLE déclare une relation et qu'une instruction ALTER TABLE ajoute des colonnes, nous ne capturons pas cela. Pourtant, alors que les instructions ALTER TABLE apparaissent 489 fois, aucune d'entre elles n'introduit des types complexes. C'est donc une menace qui peut être ignorée en toute sécurité.

(3) En cherchant des constructions de requête sur des types complexes, nous avons construit nos modèles avec diligence, mais nous ne pouvons pas exclure qu'il y ait des occurrences que nous avons peut-être manquées.

(4) Bien entendu, il serait souhaitable d'étudier plus les moteurs SQL sur Hadoop avec des relations de valeur complexes, tels que Impala.

Validité externe. La question fondamentale est de savoir si les études sur les projets de sources libres peuvent être généralisées à tous les environnements commerciaux. Bird et al. (2015), Kalliamvakou et al. (2014).

En outre, il est connu que les deux tiers des dépôts sur GitHub sont des projets de développement personnelKalliamvakou et al. (2014) où les développeurs expérimentent. En fait, le Tableau1 contient seulement deux projets avec plus d'un contributeur. Dans le même temps, les projets de sources libres sont facilement disponibles pour étude, contrairement aux projets internes à l'entreprise.

7 Travaux connexes

Dans la théorie des bases de données, il existe une longue tradition de recherche sur les formes normales ainsi que sur des bases de données complexes, remontant aux années 80. Abiteboul et al. (1995).

Dans la recherche en génie logiciel, l'analyse des applications de sources libres est une pratique bien établie, et nous nous référons à Bird et al. (2015) pour obtenir un aperçu. Il est naturel que la disponibilité de dossiers publiques de code ait permis des études empiriques sur les schémas de bases de données. Par exemple, il existe une série d'études sur l'évolution des schémas, e.g. Curino et al. (2008), Qiu et al. (2013).

Il existe également une série d'études empiriques sur des modèles de données imbriquées. Par exemple, il existe des études sur les schémas DTD Choi (2002) et XML Bex et al. (2004). Ces études sont très similaires dans leur méthodologie (collecte et analyse de schémas, et calcul de statistiques descriptives sur des propriétés structurelles).

Les auteurs dans Sauer et Härder (2013) fournissent une évaluation qualitative des langages d'interrogation pouvant être compilés dans des flux de travail MapReduce. En particulier, ils discutent des desiderata génériques, tels le traitement des données imbriquées. En revanche, nous effectuons une première évaluation quantitative de l'utilisation de ces fonctionnalités dans les projets basés sur Hive.

8 Conclusion et travaux futurs

En analysant les schémas Hive dans des référentiels de sources libres, nous pouvons montrer que des relations à valeurs complexes se produisent, mais dans une faible mesure seulement. Dans les cas observés, l'imbrication est peu profonde, ce qui peut être dû à diverses raisons, telles que la complexité accrue des requêtes écrites. Néanmoins, l'imbrication a lieu de manière sporadique et devrait donc être prise en compte lors de l'analyse comparative de Hive.

Dans les travaux futurs, nous prévoyons de mener une étude à plus grande échelle impliquant de nouveaux moteurs SQL sur Hadoop, afin d'obtenir une image encore plus globale.

Références

Abiteboul, S. et N. Bidoit (1984). Non First Normal Form Relations to Represent Hierarchically Organized Data. In *Proc. PODS'84*.

Abiteboul, S., R. Hull, et V. Vianu (Eds.) (1995). *Foundations of Databases : The Logical Level* (1st ed.). Addison-Wesley Longman Publishing Co., Inc.

Bex, G. J., F. Neven, et J. Van den Bussche (2004). DTDs Versus XML Schema : A Practical Study. In *Proc. WebDB'04*.

Bird, C., T. Menzies, et T. Zimmermann (2015). *The Art and Science of Analyzing Software Data* (1st ed.). Morgan Kaufmann Publishers Inc.

Capriolo, E., D. Wampler, et J. Rutherglen (2012). *Programming Hive* (1st ed.). O'Reilly Media, Inc.

Choi, B. (2002). What are real DTDs like ? In *Proc. WebDB'02*.

Cooper, B. F., A. Silberstein, E. Tam, R. Ramakrishnan, et R. Sears (2010). Benchmarking cloud serving systems with YCSB. In *Proceedings of the Symposium on Cloud Computing*, Indianapolis, Indiana, USA, pp. 143–154.

Curino, C. A., L. Tanca, H. J. Moon, et C. Zaniolo (2008). Schema evolution in Wikipedia : Toward a Web Information System Benchmark. In *Proc. ICEIS'08*.

DeWitt, D. J. (1993). The Wisconsin Benchmark : Past, Present, and Future. In *The Benchmark Handbook for Database and Transaction Systems (2nd Edition)*. Morgan Kaufmann.

Huang, S., J. Huang, J. Dai, T. Xie, et al. (2010). The HiBench benchmark suite : Characterization of the MapReduce-based data analysis. In *Proc. ICDEW'10*.

Jain, S., D. Moritz, et B. Howe (2016). High variety cloud databases. In *Proc. ICDE Workshops'16*.

Kalliamvakou, E., G. Gousios, K. Blincoe, L. Singer, et al. (2014). The Promises and Perils of Mining GitHub. In *Proc. MSR'14*.

Kornacker, M., A. Behm, V. Bittorf, T. Bobrovytsky, et al. (2015). Impala : A Modern, Open-Source SQL Engine for Hadoop. In *Proc. CIDR'15*.

Pavlo, A., E. Paulson, A. Rasin, D. J. Abadi, et al. (2009). A comparison of approaches to large-scale data analysis. In *Proc. SIGMOD'09*.

Poess, M., T. Rabl, et H. Jacobsen (2017). Analysis of TPC-DS : the first standard benchmark for SQL-based big data systems. In *Proc. SoCC'17*.

Qiu, D., B. Li, et Z. Su (2013). An Empirical Analysis of the Co-evolution of Schema and Code in Database Applications. In *Proc. ESEC/FSE'13*.

Sauer, C. et T. Härder (2013). Compilation of Query Languages into MapReduce. *Datenbank-Spektrum 13*(1), 5–15.

Shankar, D., X. Lu, M. Wasi-ur-Rahman, N. S. Islam, et D. K. Panda (2014). A Micro-benchmark Suite for Evaluating Hadoop MapReduce on High-Performance Networks. In *Proc. BPOE*.

Thusoo, A., J. S. Sarma, N. Jain, Z. Shao, et al. (2009). Hive : A Warehousing Solution over a Map-Reduce Framework. *Proc. VLDB Endow. 2*(2), 1626–1629.

White, T. (2009). *Hadoop : The Definitive Guide* (3rd ed.). O'Reilly Media, Inc.

Zhang, C., J. Lu, P. Xu, et Y. Chen (2018). : A Benchmark for Multi-model Database Management Systems. In *Proc. TPCTC'18*.

Summary

In this paper, we raise the question how data architects model their data for processing in Hive. This well-known SQL-on-Hadoop engine allows for complex value relations, where attribute types need not be atomic. In an empirical study, we analyze Hive schemas in open source repositories. We examine to which extent practitioners make use of complex value relations and accordingly, whether they write queries over complex types. Understanding which features are actively used will help us make the right decisions in setting up benchmarks for SQL-on-Hadoop engines, as well as in choosing which query operators to optimize for.