Un benchmark enrichi pour l'évaluation des entrepôts de données noSQL volumineuses et variables

Max Chevalier*, Mohammed El Malki*
Arlind Kopliku * Olivier Tete* Ronan Tournier *

*IRIT : Institut de Recherche en Informatique de Toulouse
118 Route de Narbonne, 31062 TOULOUSE
prenom.nom@irit.fr
http://www.irit.fr

Résumé. Avec le développement des données massives (Big Data), de nouveaux besoins émergent dans l'évaluation des systèmes d'information décisionnels. En particulier, les bancs d'essais (benchmarks) dédiés aux entrepôts de données multidimensionnelles doivent être adaptés aux volumes et à la diversité des données massives. Dans ce contexte, nous proposons un nouveau benchmark dédié aux entrepôts des données multidimensionnelles qui supporte plusieurs types de systèmes (relationnel, noSQL) et des modèles de données (flocon, étoile, aplati) structurées et non structurées. Pour tester des volumes très importants, il supporte la génération parallèle sur plusieurs machines (cluster). Il enrichit le processus de génération des données pour évaluer plusieurs niveaux de diversité des données. Dans ce papier, nous présentons ce benchmark, appelé KoalaBench, et les premiers résultats expérimentaux de son utilisation.

1 Introduction

Différents bancs d'essais (benchmarks) ont été proposés pour comparer des systèmes d'informations décisionnels. Ils fournissent des jeux de données et des scenarii d'utilisation permettant d'évaluer le comportement des systèmes, dans des conditions équivalentes, permettant ainsi une évaluation comparative. Dans le contexte des entrepôts de données multidimensionnelles (Chaudhuri et Dayal (1997)), les bancs d'essais les plus utilisés sont TPC-DS et TPC-H [1], (Zhang et al. (2004), Poess et al. (2007)). Ces solutions sont développées et optimisées pour les bases de données relationnelles (R-OLAP) et pour une utilisation sur une seule machine. La généralisation des technologies de l'information au travers de réseaux mondialisés, la diffusion massive de moyens de communications mobiles, et le développement d'objets autonomes connectés, produisent des quantités de données numérisées dans des proportions et avec un rythme sans commune mesure avec le passé. On désigne ce phénomène par le terme de mégadonnées ou « Big Data ». Les Big Data remettent en cause bon nombre d'approches

1. COUNCIL, Transaction Processing Performance. TPC-H (ad-hoc, decision support) benchmark. URL : http ://www. tpc. org/tpch, 2004

classiques dans les systèmes d'informations décisionnels. Ces derniers doivent faire face à d'énormes quantités de données disparates, c'est à dire, très variables, structurées ou non, parfois imparfaites (Moniruzzaman et Hossain (2013)). Dans ce contexte, nous proposons un nouveau benchmark, appelé Koalabench, dédié aux entrepôts de données multidimensionnelles utilisables sur un ensemble de machines (cluster) pour pouvoir atteindre d'importants volumes, et capable de supporter la variabilité des données. Pour comprendre le besoin de faire évoluer les bancs d'essais plusieurs critères peuvent être considérés :

1. *Nouvelles technologies*. Différents systèmes noSQL (« not only SQL ») existent de nos jours, pour faciliter la gestion de masses de données que les systèmes relationnels peinent à prendre en charge efficacement. Ces dernières permettent le stockage selon des modèles de données différents (documents, colonnes, graphes), introduisant une plus grande flexibilité au niveau des schémas. En termes de besoins d'évaluation, nous faisons face à une diversité de solutions à supporter.
2. *Besoin de plateformes multi-modèles*. Chaque solution noSQL supporte des formats et des modèles de données différents. Les entrepôts de données multidimensionnelles implantés avec des bases de données relationnelles (ROLAP) ont privilégié les modèles de données tels que le flocon de neige (schémas normalisés) ou l'étoile (schémas dénormalisés). Les systèmes noSQL privilégient le modèle à plat avec une dénormalisation totale (pouvant s'apparenter à la relation universelle) ou par imbrications (Chevalier et al. (2015a), Chevalier et al. (2015b)). Ces approches rompent avec le principe introduit par l'approche relationnelle de séparation stricte entre le modèle de données et les traitements effectués sur les données. Cette dépendance entre les données et les traitements, rend plus important d'avoir un support de plusieurs modèles de données adaptés à différents traitements (Stonebraker et al. (2007)).
3. *Volume*. Comparer les systèmes avec un volume de données important est devenu crucial. Plus le volume est important plus nous sommes confrontés aux limites de mémoire lors d'une configuration avec une seule machine. Les nouvelles solutions Big Data permettent le passage à l'échelle et pallient ces problèmes de mémoire. Les données sont réparties sur plusieurs machines formant un cluster, et lorsque la limite de stockage est atteinte, de nouvelles machines peuvent être facilement ajoutées. Cette technique est moins coûteuse qu'augmenter la capacité de stockage d'une seule machine serveur. Les bancs d'essai existants génèrent les données sur une seule machine.
4. *Variété*. Les modèles sur lesquels reposent les systèmes noSQL autorisent une plus grande flexibilité en supportant différents schémas pour un même ensemble de données (« schemaless »). L'intégration et l'analyse de ces données diversifiées est un processus complexe que les bancs d'essais dédiés aux systèmes décisionnels existants supportent mal.

Ils existent des solutions pour l'évaluation de données massives, mais peu d'efforts au regard des entrepôts de données, et notamment à l'organisation multidimensionnelle qui caractérise ces derniers (Dehdouh et al. (2014), Chevalier et al. (2015c)). Dans le contexte des entrepôts de données multidimensionnelles et l'analyse OLAP, plusieurs solutions existent telles que TPC-DS (Poess et al. (2007)), TPC-H (Zhang et al. (2004)) et SSB O'Neil et al. (2009), mais ces solutions ne sont définies ni pour une utilisation dans un environnement distribué, ni pour des bases de données noSQL. Leur processus de génération de données est relativement

sophistiqué et intéressant mais elles nécessitent beaucoup plus de temps lorsqu'il est question dévaluer un important volume de données (téraoctet et plus). Une autre limite importante est qu'il n'est pas prévu de supporter des données à structures variables.

Dans ce contexte, et en l'absence d'un benchmark décisionnel conçu pour les systèmes noSQL avec une prise en charge efficace du volume et de la variabilité des données, nous proposons une extension du TPC-H, un banc d'essai très populaire dans le contexte de laide à la décision (DSS). Nous étendons les fonctionnalités du système à la génération distribuée des données au niveau du système de fichiers distribués Hadoop Data File System, HDFS (Lee et al. (2012)). Les données peuvent être générées en mode normalisé (adapté aux bases de données relationnelles) mais également en mode dénormalisé (adapté aux systèmes noSQL) avec un schéma fixe ou un schéma variable. Les données peuvent être générées en différents formats (non exclusivement csv). Générer des données directement dans un format spécifique peut s'avérer intéressant pour les solutions noSQL dont le processus de chargement est facilité par un format approprié ; par exemple MongoDB privilégie les fichiers au format json (Dede et al. (2013)).

Les nouvelles fonctionnalités de ce banc d'essai, dénommé KoalaBench, se résument comme suit :

— génération de données pour différents systèmes de stockage relationnel et noSQL ;
— génération parallèle et distribuée des données nativement dans HDFS ;
— génération conforme à plusieurs modèles de données sont disponibles ;
— génération de données conformes à un schéma fixe ou variables suivant des schémas variables (schéma dynamique).

Cet article est organisé comme suit. La section 2 détaille l'état de lart des benchmarks existants. La section 3 décrit notre proposition de benchmark. La section 4 présente nos premières d'expérimentations avec cet outil.

2 Etat de l'art

Il existe de nombreux travaux dans le domaine du développement des bancs d'essai dédiés à la gestion de données et aux systèmes dinformations décisionnels. L'évolution et le développement des systèmes de stockage suscitent une évolution continue des bancs d'essai. Nous distinguons deux familles de bancs d'essai, ceux utilisés dans le contexte d'aide à la décision et ceux dédiés au contexte Big Data. En ce qui concerne la première famille, nous détaillons les méthodes basées sur les bancs d'essai TPC qui concernent les systèmes d'aide à la décision. Quant à la deuxième famille d'approches, nous détaillons des bancs d'essai qui supportent les systèmes distribués.

2.1 Les bancs d'essai décisionnels (DSS)

Les bancs d'essai dédiés aux systèmes décisionnels, DSS (Decision Support System) sont édités par TPC (Transaction Processing Council). Ils représentent les systèmes les plus utilisés pour évaluer les performances des bancs d'essai DSS. Le premier banc d'essai est l'APB-1 l'argement exploité dans les années 90 mais devenu obsolète et inadapté pour la plupart des cas d'évaluation expérimentale (Darmont (2009)). Par la suite, le banc d'essai TPC-D a été proposé. C'est le système à partir duquel ont été dérivés les deux systèmes TPC-H et TPC-R.

Le modèle de TPC-H est plus riche, normalisé et permet de supporter une centaine de requêtes, classées en 4 catégories : requêtes OLAP interactives, requêtes d'aide à la décision ad-hoc, requêtes d'extraction et requêtes de rapports. Les données sont structurées conformément à un schéma fixe en flocon. En 2009, un autre banc d'essai avec un schéma en étoile (dénormalisation des dimensions) a été proposé. Il s'agit de SSB (Start Schema Benchmark). Il introduit un certain niveau de dénormalisation des données pour des raisons de simplicité. Il modélise un schéma en étoile composé d'un unique fait et quatre dimensions. Nous rencontrons dans ce contexte, deux des rares efforts pour adapter le benchmark SSB aux systèmes noSQL, il s'agit du banc d'essai CNSSB (Dehdouh et al. (2014)) et le banc d'essai SBB+ (Chevalier et al. (2015c)). Le CNSSB supporte un modèle de données orientés colonnes et propose deux modèles logiques pour structurer le modèle conceptuel en étoile. Le banc d'essai SBB+ est adapté pour les modèles orientés colonnes et orientés documents, et offre 4 modèles logiques pour chacun des systèmes noSQL supportés. Ces deux dernières solutions reposent sur un jeu de requêtes simples contrairement à TPC qui offre un jeu de requêtes plus riche avec un degré de complexité plus important. De plus, elles se limitent à la génération de données fortement structurées, conformes à un schéma fixe, sans support de données variables au niveau du fait et des dimensions. Les bancs d'essai TPC demeurent la référence la plus utilisée pour les systèmes d'aide à la décision. Ils sont bien adaptés aux systèmes R-OLAP mais ne peuvent pas être facilement mis en œuvre avec des bases de données noSQL en cluster et avec une approche par schémas dynamiques (variabilité des données).

2.2 Les bancs d'essai Big Data

Les bancs d'essai dédiés aux Big Data visent à comparer les nouveaux systèmes qui stockent des données massivement distribuées et supportent des calculs parallèles. Le service Yahoo Cloud est l'un des outils les plus populaires (Cooper et al. (2010)). Il est utilisé pour comparer les opérations CRUD élémentaires (Create, Read, Update et Delete). Il a déjà été exploité pour la plupart des systèmes noSQL et offre de bonnes performances en termes de chargement de données. De la même manière, Bigframe est un banc d'essai qui se focalise principalement sur les problèmes de volume, de variété et de vélocité dans le contexte Big Data (Big). Avec plus de fonctionnalités, BigBench propose un cadre d'évaluation plus riche que les deux premiers (Ghazal et al. (2013)). Ce dernier modélise des lignes de commandes. Il comprend 3 types de données : structurées (issues du TPC-DS), semi-structurées (flux de clics dans les sites web), et non structurées (commentaires de clients). Contrairement aux bancs d'essai traditionnels, les bancs d'essai Big Data sont orientés sur la flexibilité de l'information, des données massives et évolutives et n'évaluent pas les mêmes critères que les bancs d'essai DSS.

D'autres bancs d'essai ont été aussi proposés. HadoopToSQL (Iu et Zwaenepoel (2010)), évalue les performances de MapReduce pour la charge de travail. Les requêtes MapReduce sont transformées pour utiliser l'indexation, le regroupement et l'agrégation des attributs fournis par les bases de données SQL. De la même façon, dans les travaux de (Lee et al. (2011)), les auteurs proposent la solution YSmart, un banc d'essai construit au-dessus de la plate-forme Hadoop qui permet de traduire des requêtes SQL en requêtes MapReduce. La traduction est assurée via un ensemble de fonctions MapReduce. Dans les travaux de (Moussa (2012)), l'auteur définit un ensemble de règles pour traduire le jeu de requêtes TPC-H de SQL en Pig Latin. Il propose une approche de construction des requêtes en Pig pour maximiser les performances.

3 Le banc d'essai KoalaBench

Le banc d'essai KoalaBench est conçu pour répondre aux besoins des systèmes décisionnels basés sur des entrepôts de données multidimensionnelles massives. Il est dérivé du banc d'essai TPC-H, le banc d'essai de référence, pour évaluer les systèmes décisionnels. Le générateur de données est développé en Java à partir du TPC-H. Il supporte :

— des modèles logiques différents (plat, étoile, flocon et plat flexible) ;
— des formats multiples compatibles avec les systèmes de stockage relationnel et noSQL ;
— une génération parallèle des données avec le système de fichiers distribués HDFS.

3.1 Modèle de données

La nouvelle extension enrichit le banc d'essai de base et permet à l'utilisateur de choisir le modèle de données approprié lors du processus de génération. En effet, les modèles noSQL se caractérisent par une forte dépendance au traitement : l'efficacité d'une structure de données dépend du type d'analyses appliquées (contrairement à la modélisation relationnelle dont les formes normales sont définies indépendamment des traitements). Ainsi, suivant le type d'analyse, le choix de structuration peut être différent. Il est même parfois nécessaire de concevoir un entrepôt de données combinant deux implantations logiques ou plus, chacune favorisant un type d'analyse. De plus, les données à manipuler sont de moins en moins homogènes, avec un schéma pouvant varier d'un enregistrement à l'autre. Désormais, il devient important d'avoir des modèles de données qui considèrent la diversité des schémas. A cet effet, l'extension que nous proposons permet de générer plusieurs modèles de données que nous classons en deux catégories de modèles : modèles fortement structurées et modèles flexibles.

Plus précisément, le banc d'essai supporte les modèles suivants :

— Schéma en flocon : ceci est le modèle de données référencé sur le banc TPC-H ; il est préféré pour les entrepôts de données implantés sur des bases de données relationnelles.
— Schéma en étoile : ce modèle est une simplification du modèle flocon. Nous utilisons le même que celui utilisé dans SSB. Il est également adapté pour une utilisation avec des bases de données relationnelles.
— Schéma plat : il est préféré par la majorité des systèmes noSQL, en évitant les calculs de jointures.

3.1.1 Modèle de données en flocon

Ce premier modèle de données est très proche de celui proposé dans le banc d'essai TPC -H avec des modifications mineures. Il correspond aux modèles de données en flocon. La redondance des données est très réduite ; par exemple, un client peut référencer un pays référençant à son tour une région (Europe). Les *régions* et les *pays* sont générés dans des fichiers différents. Le schéma de données est représenté sur la Fig 1. Dans ce modèle, les données sont générées dans 9 fichiers qui serviront à alimenter la base de données.

3.1.2 Modèle de données en étoile

Ce modèle de données correspond au modèle de données en étoile utilisé dans le benchmark SSB. Il est dérivé du modèle de données en flocon, mais il est simplifié pour considérer

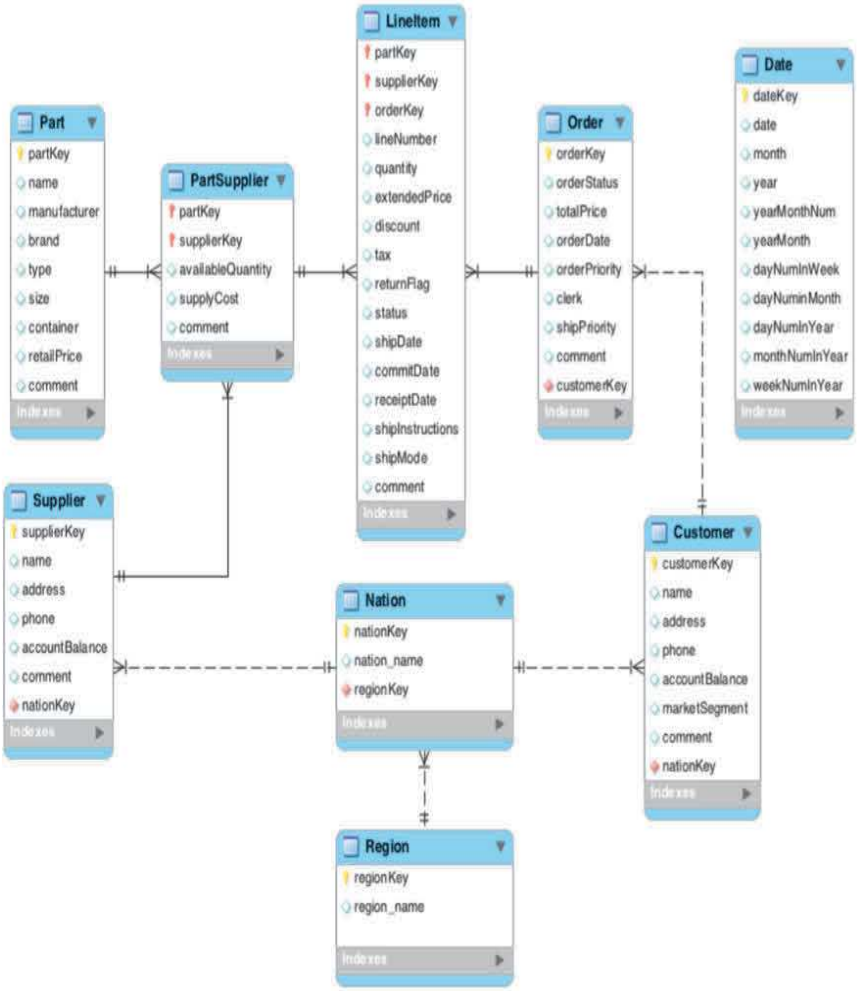

FIG. 1 – *Modèle de données flocon de neige*

une seule entité pour le fait et une seule entité par dimension (Ravat et al. (2009), Morfonios et al. (2007)). Les données des dimensions sont ainsi dénormalisées, comportat alors des redondances (les données fonctionnellement dépendantes) ; par exemple, un client est associé à un pays, lui-même associé à une région. Dans le modèle en étoile, nous ne générons plus qu'un seul fichier pour les données du pays et les données de la région, en répétant le pays pour chaque région qui lui correspond.

Nous ajoutons aussi une nouvelle table « date ». La date devient une dimension explicite, décrite selon plusieurs attributs tels que «week number», «day of week» et «day in year». Au final, avec ce modèle, un fichier plat est généré par concept multidimensionnel (fait et

dimensions) avec un total de 5 fichiers.

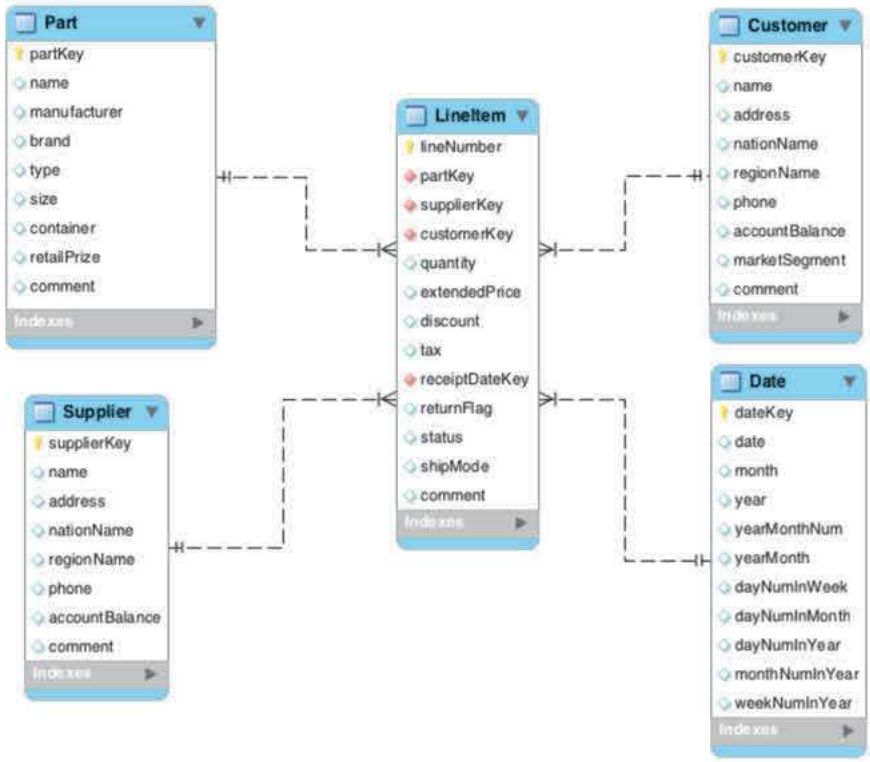

FIG. 2 – *Modèle de données en étoile*

3.1.3 Modèle de données plat

Ce modèle de données favorise la dénormalisation complète des données en regroupant le fait et les dimensions dans une seule structure. Il est adapté aux bases de données noSQL qui supportent difficilement les jointures. Par conséquent, il induit une importante redondance des données des dimensions. Avec ce modèle, un seul fichier est généré regroupant les mesures et attributs des dimensions.

3.1.4 Diversité des schémas

KoalaBench permet à l'utilisateur de générer des données avec des schémas hétérogènes (schéma flexible). Cette diversité de données est spécifique aux bases de données noSQL car il considère la variabilité, autrement dit, un relâchement au niveau de la structure du schéma. Désormais dans cette nouvelle version, l'utilisateur peut évaluer la volumétrie des données mais aussi la diversité des schémas. Ainsi, nous ajoutons un fichier de configuration du générateur des données permettant à l'utilisateur de choisir deux options :

— diversity : indique le nombre de classes de schémas possibles. Les schémas d'une même classe sont homogènes.
— homogeneity : indique la répartition des données en fonction de la diversité, c'est à dire le nombre d'enregistrements par classe de schémas.

Pour illustrer, considérons *A* l'ensemble de tous les attributs tel que $A = [a_1, a_2, ..., a_n]$ et *T* l'ensemble des enregistrements tel que $T = [t_1, t_2, ..., t_m]$. Avec un degré de diversité égal à 2, nous obtenons deux classes de schémas :

— $C_{S1} = [b_1, b_2, ..., b_p]$ un ensemble d'attributs, $\forall j \in [1..p], b_j \in A$
— $C_{S2} = [c_1, c_2, ..., c_q]$ un ensemble d'attributs, $\forall j \in [1..q], c_j \in A$

Le choix des attributs est déterminé par l'utilisateur, dans un fichier de configuration des schémas et de leur distribution ; $C_{S1} \cup C_{S2} = A$. L'utilisateur détermine depuis ce fichier les schémas et le pourcentage des distributions. L'homogénéité spécifie la distribution des données pour les différents schémas possibles ; le degré d'homogénéité est proportionnel au degré de la diversité. Par exemple, avec un degré de diversité à 2, l'homogénéité est égale à $\frac{1}{2} + \frac{1}{2}$, formant deux ensembles de données tels que $T1 \cup T2 = T$ et $T1 \cap T2 = \emptyset$:

— T_1 : 50% d'enregistrements avec un schéma de données de classe C_{S1}.
— T_2 : 50% d'enregistrements avec un schéma de données de classe C_{S2}.

Remarque 1 : Dans cette extension la flexibilité ne concerne pas les attributs racines (identifiant) des dimensions.

Remarque 2 : L'absence de certains attributs dans les schémas de données qui sont potentiellement divers, a une conséquence directe sur les résultats et l'utilisation des requêtes. Certaines requêtes peuvent devenir invalides, si les attributs qui apparaissent en elles n'apparaissent dans aucun des schémas. Pour répondre à cette problématique, nous recommandons le choix de classes de schémas qui couvrent tous les attributs $C_{S1} \cup C_{S2} = A$

3.2 Formats de fichiers et systèmes supportés

Plusieurs formats de fichiers sont possibles : tbl, csv, json et xml. Afin d'optimiser la phase de chargements des données dans les systèmes noSQL, le générateur offre à l'utilisateur la possibilité de spécifier le format approprié au système noSQL utilisé. Par exemple, dans le modèle orienté documents, MongoDB est un système qui stocke les données en bjson (binary json), il est optimisée pour un chargement à partir de fichiers json. Charger un fichier de format csv dans MongoDB est possible mais nécessitera une conversion en json qui augmente considérablement le temps de chargement.

En fonction des formats et modèles de données générés, plusieurs gestionnaires de base de données sont supportés.

— bases de données relationnelles : PostgreSQL, MySQL, Oracle qui acceptent des fichiers sous format csv, tbl ou xml.
— bases de données XML ou orienté objet qui acceptent des fichiers du format xml.
— bases de données orientés documents : MongoDB, CouchDB, qui acceptent des fichiers du format json.
— bases de données orientés colonnes : HBase, Cassandra qui acceptent des fichiers au format csv.
— bases de données orientées graphes : Neo4j qui acceptent des fichiers au format csv.

3.3 Génération massivement distribuée

Le générateur de données a été adapté pour générer les données sur plusieurs machines en même parallèle. Il est possible de générer les données sur le système de fichiers distribués HDFS qui constitue un système de fichiers très populaire. La génération distribuée se base sur les deux principaux composants du système de fichiers distribués Hadoop :

— HDFS : est utilisé pour stocker le résultat de chaque tâche Map.

— MapReduce : assure la distribution du processus de génération des données.

Notons que nous avons uniquement besoin de la phase Map, la phase Reduce nest pas nécessaire dans la distribution de la génération des données.

3.4 Impact de l'indicateur d'échelle

Dans cette section, nous expliquerons l'impact de l'indicateur d'échelle (*sf*) sur le volume des données générées. L'indicateur d'échelle permet de spécifier le volume de données à générer. Par exemple pour générer un volume de données de 10GB il convient d'utiliser un indicateur d'échelle *sf = 10*. Nous discutons ces détails pour un*sf=1*, pour les autres indicateurs d'échelle, les proportions sont quasi-linéaires.

Flocon. Nous générons 8 fichiers, à raison d'un fichier par entité. Nous obtenons 5247925 line items, 1500000 orders, 150000 customers, 800000 supplied parts, 200000 parts, 10000 suppliers, 25 nations, 5 regions. Le total de l'espace de stockage nécessaire dépend du format de fichiers, plus précisément, cela nécessite 3.87GB en xml, 2.33GB en json, 1.16GB en tbl et csv.

Star. Nous générons 5 fichiers, à raison d'un fichier par entité. Nous obtenons 5247925 line items, 150000 customers, 200000 parts, 10000 suppliers et 2556 dates. En terme d'espace de stockage, nous avons 2.52GB en xml, 1.47GB en json, 0.68GB en tbl et csv.

Plat : Nous générons 1 seul fichier. Nous obtenons un total de 5247925 line items. En terme d'espace de stockage, nous avons 8.21GB en xml, 4.67GB en json, 2.38GB en tbl et csv.

L'ensemble des résultats est résumé dans le tableau 1 :

Modèle/Format	.xml	.json	.tbl	.csv
Flocon	3.87 GB	2.33 GB	1.16 GB	1.16 GB
Etoile	2.52 GB	1.47 GB	0.68 GB	0.68 GB
Plat	8.21 GB	4.67 GB	2.38 GB	2.38 GB
Plat Flexible	6.48 GB	4.12 GB	2.13 GB	2.13 GB

TAB. 1 – *Taille des fichiers par modéle*

3.5 Le jeu de requêtes

Nous conservons le générateur QGEN d'origine. Les requêtes générées sont écrites en SQL. Nous ne fournissons pas à l'heure actuelle la traduction des requêtes selon différents systèmes car les systèmes noSQL et leur langages d'interrogation sont spécifiques et en constante évolution. Le générateur de requêtes a un nombre important de requêtes mais certaines ne

peuvent pas être traduites dans un langage d'interrogation noSQL du fait qu'elles ne sont pas adaptées pour les modèles de données dénormalisées ou que certains critères n'ont pas à être évalués tels que les jointures. Ce travail d'adaptation reste donc à la charge des équipes d'utilisateurs de KoalaBench.

Pour nos propres expériences, nous avons réécrit les requêtes pour les adapter aux modèles de données dénormalisées et les avons traduites dans les langages des bases de données noSQL cibles : Hive pour une utilisation avec HBase, CQL pour une utilisation avec *Cassandra*, dans le langage de requête *MongoDB* pour une utilisation avec *MongoDB* et dans le langage *Cypher* pour une utilisation avec *Neo4j*. Ces requêtes peuvent être classées selon deux critères :

— La dimensionnalité affecte le nombre de dimensions dans les clauses de regroupement (équivalent à la clause Group By en SQL) : 1D pour une dimension, 2D pour deux dimensions, 3D pour trois dimensions et 4D pour quatre dimensions.
— La sélectivité affecte le degré de filtrage des données quand des conditions sont appliquées.

4 Expérimentations

Dans cette section, nous présentons les résultats des premières expériences menées avec KoalaBench :

— comparaison de l'espace mémoire disque ;
— comparaison du temps de génération des données ;
— comparaison du temps de chargement dans les systèmes Cassandra et MongoDB.

Pour chaque configuration, nous faisons varier l'indicateur d'échelle pour faire des comparaisons avec différents volumes.

Environnement de test. Pour réaliser ces expérimentations nous avons mis en place un environnement distribué basé sur la plateforme Hadoop 2.7.2 et le système de gestion de données noSQL orienté colonnes Cassandra 3.0 et orienté documents MongoDB 3.2. L'architecture distribuée est un cluster composé de trois nœuds. Chaque nœud est une machine Unix (Centos 7) avec 4 Core-i5 et une mémoire RAM de 8Go. Chaque nœud dispose d'un espace de stockage de 2x2To et une connexion réseau de 1Gb/s. Chaque nœud est utilisé pour jouer le rôle de datanode et l'un deux joue en plus le rôle de namenode. Pour le système de gestion orienté documents, nous avons installé MongoDB au niveau du cluster. Nous avons installé MongoDB v3.2 dans chaque nœud. Dans la terminologie MongoDB, cette configuration correspond à trois shards (un par machine) et l'un deux agit comme maître, gérant la répartition et les traitements (shardings). Pour le système de gestion orienté colonnes, nous avons installé une instance Cassandra sur chacun des trois nœuds. Contrairement à MongoDB qui repose sur une communication «maître-esclave», l'architecture de Cassandra repose sur une communication « maître-maître ». Nous adoptons ces deux systèmes parce qu'il offrent :

— deux architectures différentes de communication, maître-maître pour Cassandra et maître-esclave pour MongoDB ;
— un outil de chargement intégré facilitant le chargement de données importantes (en bulk) ;
— un outil de supervision permettant de suivre le chargement.

Expérimentation 1 : Utilisation de la mémoire.

Dans le Tab 2, nous reportons le temps nécessaire pour générer les données en utilisant différents indicateurs d'échelle (*sf=1, sf=10, sf=100 et sf=1000*) pour les différents modèles de données. Le format du fichier impacte significativement la mémoire disque, comme nous pouvons l'observer dans le Tab 3 ; par exemple pour un *sf=1000*, nous avons, selon le modèle plat, 2380GB pour le format csv et 4670GB pour le format json, soit environ le double. Cette différence impacte significativement le temps de génération, puisque les fichiers avec un format expressif (json ou xml) nécessitent de 3 à 4 fois plus de temps ; par exemple pour générer un sf=1000, selon le modèle plat, nous avons besoin de 17883 secondes pour un fichier csv contre 69872 secondes pour un fichier json. Une autre observation peut être faite entre la génération fixe et flexible sur le modèle à plat. On remarque que la génération flexible est plus rapide, ce qui s'explique par la moindre volumétrie générée.

		sf1	sf10	sf100	sf1000
Flocon	xml	50.2s	386s	3634s	35902s
	json	39.2s	298s	2873	27453s
	csv	23.7s	173s	1877	17832s
	tbl	23.7s	173s	1877	17832s
Etoile	xml	30.5s	192s	2028	18973s
	json	40.5s	244s	2351	21839
	csv	20.8s	123s	1312s	12893s
	tbl	20.8s	123s	1312s	12893s
Plat flexible	xml	122s	1143s	11165s	104321s
	json	71s	561s	5835s	56348s
	csv	21s	136s	1560s	14902s
	tbl	21s	136s	1560s	14902s
Plat	xml	154s	1372s	13767	132756s
	json	87s	691s	7003s	69872s
	csv	31s	164s	1873s	17883s
	tbl	31s	164s	1873s	17883s

TAB. 2 – *Temps de génération par modèle et par indicateur d'échelle*

Expérimentation 2 : Temps de chargement

Dans le Tab 4, nous reportons le temps nécessaire pour charger les données en utilisant différents indicateurs d'échelle (*sf=1, sf=10, sf=100*) pour les différents modèles de données. Nous avons procédé uniquement au chargement de format approprié pour chaque outil, soit json pour MongoDB et csv pour Cassandra. Nous pouvons constater que le temps de chargement nécessaire est moins important dans Cassandra que dans MongoDB ; par exemple, pour un *sf=1*, nous avons besoin de 672 secondes pour Cassandra contre 3967 secondes pour MongoDB soit plus de 4 fois. Cela s'explique par le format de fichier utilisé. Dans MongoDB, le format json utilisé est 4 fois plus volumineux qu'un fichier csv. En outre, cela engendre plus de transfert entre le nœud maître et le nœud esclave. La communication dans Cassandra via son architecture maître-maître semble moins coûteuse. MongoDB crée un nombre important d'index pour pouvoir optimiser la phase d'interrogation.

		sf1	sf10	sf100	sf1000
Flocon	xml	3.87 GB	38.7 GB	387 GB	3870 GB
	json	2.33 GB	23.3 GB	233 GB	2330 GB
	csv	1.16 GB	11.6 GB	116 GB	1160 GB
	tbl	1.16 GB	11.6 GB	116 GB	1160 GB
Etoile	xml	2.52 GB	25.2 GB	252 GB	2520 GB
	json	1.47 GB	14.7 GB	147 GB	1470 GB
	csv	0.68 GB	6.8 GB	68 GB	680 GB
	tbl	0.68 GB	6.8 GB	68 GB	680 GB
Plat flexible	xml	6.48 GB	64.8 GB	64.8 GB	6480 GB
	json	4.12 GB	41.2 GB	412 GB	4120 GB
	csv	2.13 GB	21.3 GB	213 GB	2130 GB
	tbl	2.13 GB	21.3 GB	213 GB	2130 GB
Plat	xml	8.21 GB	82.1 GB	821 GB	8210 GB
	json	4.67 GB	46.7 GB	467 GB	4670 GB
	csv	2.38 GB	23.8 GB	238 GB	2380 GB
	tbl	2.38 GB	23.8 GB	238 GB	2380 GB

TAB. 3 – *espace mémoire (en GB) par modèle et par indicateur d'échelle.*

	sf=1	sf=10	sf=100
Cassandra (Etoile)	672s	6643s	69025s
MongoDB (Plat)	3967s	38632s	381142s

TAB. 4 – *Temps de chargements par modèle et par indicateurs dans les deux systémes Cassandra et MongoDB (en secondes)*

5 Conclusion

Cet article présente le benchmark KoalaBench conçu pour répondre aux besoins des systèmes décisionnels basés sur des entrepôts de données multidimensionnelles massives (Big Data). Il repose sur une extension du benchmark de référence TPC-H. Cette extension réside dans sa capacité à fonctionner en mode distribué pour atteindre des volumétries massives, et sa capacité d'engendrer des données variables au sein des faits et des dimensions. Les données peuvent être générées dans différents formats (tbl, csv, xml, json) et dans différents modèles de données. KoalaBench n'est pas restreint au modèle relationnel, il peut générer les données dans plusieurs systèmes noSQL. Ce nouveau benchmark est adapté aux systèmes noSQL orientées colonnes, orientées graphes et orientées documents. Les données peuvent être générées avec un schéma flexible ou fixe, dans une architecture distribuée utilisant la plateforme Hadoop. Ce nouveau benchmark propose également un script de chargement spécifique à chaque système évalué. Nos expérimentations montrent que KoalaBench apporte de nombreux avantages par rapport à la version d'origine TPC-H. Il facilite la phase de chargement en permettant le chargement des données dans un environnement distribué Hadoop. Il permet également d'évaluer

la diversité des schémas, très spécifique aux approches noSQL. Cette fonctionnalité permet d'introduire une option permettant la génération des données avec un schéma dynamique.

Annexe

Instructions d'utilisation

Le benchmark KoalaBench est écrit en java et nécessite les librairies suivantes :

— guava-18.0

— hadoop commons

Nous recommandons de l'ouvrir avec Eclipse en important le dossier KoalaBech avec Maven. Il est aussi possible de le lancer depuis une ligne de commande mais après l'avoir compilé avec Maven. La génération des données se fait en appelant la classe DBGen comme suit : java DBGen Par défaut, les données sont générées suivant le format tbl et suivant le modèle en flocon avec l'indicateur d'échelle *sf=1*. Dans ce qui suit, nous listons les options possibles :

— Format : permet de spécifier le type de format de données, les valeurs possibles sont csv, json, xml ou tbl.

— Modèle de données : permet de spécifier un modèle de données, les valeurs possibles sont : plat, flocon, flexible et star.

— Répertoire de stockage : permet de spécifier le répertoire de stockage des données à générer. Le chemin absolue doit être indiqué précédé du symbole ">", par exemple : $> /usr/local/data/tmp\_dir/$.

— Répertoire HDFS : permet de spécifier le chemin d'accès au répertoire de stockage HDFS précédé de hdfs sans espace ($hdfs://nn1.example.com/user/hadoop/dir$)

— Diversité : permet de spécifier le nombre de classes de schémas. Les valeurs possibles sont comprises entre 1 et 10. La valeur doit être précédée de dv (exemple : dv 2).

— Homogénéité : permet de spécifier la proportion des enregistrements par classe de schémas. Les valeurs possibles sont comprises entre 1 et 100 et représente le pourcentage de chaque distribution. Le nombre de valeurs est égal au degré de diversité (exemple pour un dv = 2, peut avoir h 75-25, soit 75 % pour la première distribution et 25% pour la seconde).

Par exemple :

— java DBGen json flat sf25 : génère des données suivant le modèle plat en format json avec un sf=25.

— java DBGen json flat sf25 dv 2 h 50-50 : génère des données selon le plat en format json avec un sf=25. Les enregistrements sont divisés en deux proportions chacune avec un schéma.

— java DBGen snow sf10 csv : génère des données suivant le modèle en flocon avec un format csv L'ordre des paramètres n'est pas important.

Références

Bigframe user guide, 2013 - https ://github.com/bigframeteam/bigframe/wiki/.

Chaudhuri, S. et U. Dayal (1997). An overview of data warehousing and olap technology. *SIGMOD Rec.*, 65–74.

Chevalier, M., M. El Malki, A. Kopliku, O. Teste, et R. Tournier (2015a). Implantation not only sql des bases de données multidimensionnelles. In *4eme Seminaire de Veille Strategique, Scientifique et Technologique (VSST 2015)*, pp. 0.

Chevalier, M., M. El Malki, A. Kopliku, O. Teste, et R. Tournier (2015b). Implementation of multidimensional databases in column-oriented nosql systems. In *East European Conference on Advances in Databases and Information Systems*, pp. 79–91. Springer International Publishing.

Chevalier, M., M. E. Malki, A. Kopliku, O. Teste, et R. Tournier (2015c). Benchmark for olap on nosql technologies comparing nosql multidimensional data warehousing solutions. In *2015 IEEE 9th International Conference on Research Challenges in Information Science (RCIS)*, pp. 480–485.

Cooper, B. F., A. Silberstein, E. Tam, R. Ramakrishnan, et R. Sears (2010). Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, pp. 143–154.

Darmont, J. (2009). *Data warehouse benchmarking with DWEB*, Volume 3 of *Advances in Data Warehousing and Mining*, pp. 302–323.

Dede, E., M. Govindaraju, D. Gunter, R. S. Canon, et L. Ramakrishnan (2013). Performance evaluation of a mongodb and hadoop platform for scientific data analysis. In *Proceedings of the 4th ACM Workshop on Scientific Cloud Computing*, pp. 13–20.

Dehdouh, K., O. Boussaid, et F. Bentayeb (2014). *Columnar NoSQL Star Schema Benchmark*.

Ghazal, A., T. Rabl, M. Hu, F. Raab, M. Poess, A. Crolotte, et H.-A. Jacobsen (2013). Bigbench : Towards an industry standard benchmark for big data analytics. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13.

Iu, M.-Y. et W. Zwaenepoel (2010). Hadooptosql : A mapreduce query optimizer. In *Proceedings of the 5th European Conference on Computer Systems*, EuroSys '10.

Lee, K.-H., Y.-J. Lee, H. Choi, Y. D. Chung, et B. Moon (2012). Parallel data processing with mapreduce : A survey. *SIGMOD Rec. 40*(4).

Lee, R., T. Luo, Y. Huai, F. Wang, Y. He, et X. Zhang (2011). Ysmart : Yet another sql-to-mapreduce translator. In *2011 31st International Conference on Distributed Computing Systems*.

Moniruzzaman, A. B. M. et S. A. Hossain (2013). Nosql database : New era of databases for big data analytics - classification, characteristics and comparison. *CoRR abs/1307.0191*.

Morfonios, K., S. Konakas, Y. Ioannidis, et N. Kotsis (2007). Rolap implementations of the data cube. *ACM Comput. Surv. 39*(4).

Moussa, R. (2012). Tpc-h benchmarking of pig latin on a hadoop cluster. In *Communications and Information Technology (ICCIT), 2012 International Conference on*, pp. 85–90.

O'Neil, P. E., E. J. O'Neil, X. Chen, et S. Revilak (2009). The star schema benchmark and augmented fact table indexing. In *Performance Evaluation and Benchmarking, First TPC Technology Conference, TPCTC 2009, Lyon, France, August 24-28, 2009, Revised Selected Papers*, pp. 237–252.

Poess, M., R. O. Nambiar, et D. Walrath (2007). Why you should run tpc-ds : A workload analysis. In *Proceedings of the 33rd International Conference on Very Large Data Bases*,

VLDB '07.

Ravat, F., O. Teste, R. Tournier, et G. Zurfluh (2009). Algebraic and graphic languages for olap manipulations. *Strategic Advancements in Utilizing Data Mining and Warehousing Technologies : New Concepts and Developments : New Concepts and Developments*, 60.

Stonebraker, M., S. Madden, D. J. Abadi, S. Harizopoulos, N. Hachem, et P. Helland (2007). The end of an architectural era : (it's time for a complete rewrite). In *Proceedings of the 33rd International Conference on Very Large Data Bases*, VLDB '07.

Zhang, J., A. Sivasubramaniam, H. Franke, N. Gautam, Y. Zhang, et S. Nagar (2004). Synthesizing representative i/o workloads for tpc-h. In *Software, IEE Proceedings-*, pp. 142–142.

Summary

With the advent of Big Data technologies, there is a need for new benchmarks to evaluate information decision systems. In particular, existing benchmarks for multidimensional data warehouses need to be adapted to increasing volume and diversity of data. In this context, we propose a new benchmark dedicated to data warehouses that can support different information systems (relational and NoSQL) and data models (snowflake, star, flat) structured or non structured. In order to scale in volume, it supports parallel generation of data on multiple computers (cluster). It can generate diverse data structures, by supporting generation of multiple different schemas. In this paper, we present a new benchmark for Big data, called KoalaBench and the first experimental results of its use.