Le génie pour l'industrie

RAPPORT TECHNIQUE
PRÉSENTÉ À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
DANS LE CADRE DU COURS GTI792 PROJET DE FIN D'ÉTUDES EN GÉNIE DES TI

BASE DE DONNÉES DISTRIBUÉE APPLIQUÉE À LA GÉNÉTIQUE DANS LE CADRE DE L'ANALYSE DU SÉQUENÇAGE GÉNOMIQUE

JEAN-FRANÇOIS HAMELIN
HAMJ12068802

DÉPARTEMENT DE GÉNIE LOGICIEL ET DES TI

Professeur-superviseur

Alain April

MONTRÉAL, 12 AOÛT 2012
ÉTÉ 2012

REMERCIEMENTS

Alain April : Professeur de génie logiciel.

Patrice Dion : Analyste des systèmes et réseaux informatiques, département de systèmes éducationnels et de recherche de l'ÉTS.

Anna Klos : Diplômée de l'ÉTS en génie logiciel.

Ousmane Diallo, B.Sc. : programmeur pour le projet S2D, laboratoire Guy Rouleau, CRCHUM.

J'aimerais aussi exprimer par-dessus tout mes plus sincères remerciements à Jean-Philippe Bond, partenaire de travail de longue date et coéquipier dans ce projet, avec qui la rigueur et le souci de l'excellence sont toujours mis de l'avant.

OPTIMISATION DE RECHERCHE GRÂCE À HBASE SOUS HADOOP

JEAN-FRANÇOIS HAMELIN
HAMJ12068802

RÉSUMÉ

Ce projet s'insère dans un contexte d'affaires où le Centre de Recherche du Centre Hospitalier de l'Université de Montréal (CRCHUM) est aux prises avec des problèmes dans un système d'identification de gènes et où l'ÉTS est désireuse d'amasser du matériel en vue d'un cours sur le « big data ».

Le CRCHUM, possédant bien au-delà de 150 millions d'enregistrements de données génomiques, utilise à l'heure actuelle un système permettant d'effectuer des recherches sur des gênes afin de, par exemple, trouver certaines variantes de gènes partageant des similarités. Selon les informations publiées sur le site Web du laboratoire Roueleau, « l'objectif principal du projet Synapse to Disease (de la synapse à la maladie ou S2D) est d'identifier des gènes causant ou prédisposant à des maladies du développement et du fonctionnement neuronal » (Laboratoire Guy Rouleau, 2012). S'appuyant sur une base de données relationnelle conventionnelle, le CRCHUM voit rapidement sa solution atteindre un plateau. En effet, plusieurs de leurs requêtes sont longues à effectuer et ont déjà demandé un remaniement de la base de données important. Leurs responsables voient donc, à l'horizon, un problème dans leur capacité de stocker et effectuer des requêtes efficaces sur les données.

De son côté, le professeur Alain April réfléchit depuis un bon moment à créer de toutes pièces un cours portant sur le nouveau phénomène du « big data ». Lorsqu'il rencontra le Dr Rouleau, responsable du laboratoire Rouleau, lors d'une conférence, le professeur April vu immédiatement un potentiel dans le problème technologique de ce dernier.

En conséquence de ce qui précède, le projet aura des retombées sur les deux parties prenantes distinctes. D'un côté, le CRCHUM bénéficiera d'une preuve de concept qui peut servir comme prototype de solution à leurs problèmes de performance lors de requêtes sur plusieurs millions d'enregistrements. De l'autre côté, le professeur Alain April et son entourage pourront se servir des extrants du projet comme ressources pour monter un futur cours portant sur la technologie « big data ».

TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

API	Application Programming Interface
ADN	Acide désoxyribonucléique
CRCHUM	Centre de Recherche du CHUM
CPU	Central Processing Unit
GIG	GigaByte
HDFS	Hadoop Distributed File system
JPS	Java Process Status tool
MB	MegaByte
NoSQL	Not only SQL
RAM	Random Access Memory
SGBD	Système de gestion de base de données
SSH	Secure SHell

INTRODUCTION

Comme le quotidien La Presse le soulignait dans son article du 22 mai 2012, « l'heure est au big data » (McKenna, 2012). En effet, un nouveau phénomène dû en partie à la popularité d'Internet a pour conséquence de produire des quantités titanesques de données chaque jour. Cependant, le phénomène est bien plus large que la consommation de données sur l'Internet. En effet, les domaines comme la génomique, l'épidémiologie et ou la sécurité nationale produisent et consomment énormément de données. Bien que certaines entreprises comme Google et Facebook puissent engranger des profits monstrueux avec ces données, encore doivent-elles bien les gérer et les supporter; bienvenue à l'ère du « big data ». Comme le décrit Wikipédia, « Big data ("grosse donnée" ou données massives) est une expression anglophone utilisée pour désigner des ensembles de données qui deviennent tellement volumineux qu'ils en deviennent difficiles à travailler avec des outils classiques de gestion de base de données. Dans ces nouveaux ordres de grandeur, la capture, le stockage, la recherche, le partage, l'analyse et la visualisation des données doivent être redéfinis » (Wikipédia, 2012).

Le besoin en stockage de données à large échelle et l'engouement entourant les technologies émergentes comblant ce besoin sont tellement récents qu'aucune école spécialisée à Montréal, y compris l'ÉTS, ne dispense des cours sur la matière. Dans une toile de fond où l'ÉTS se veut chef de file en matière d'école de technologie, il est impératif de bâtir un cours ayant pour thématique le « big data ». C'est donc dans cet esprit que ce projet veut accomplir la création d'un environnement HBase distribué, bien documenté et s'appuyant sur un cadre d'utilisation concret. En effet, afin de rester le plus près possible d'une utilisation industrielle de la base de données HBase, un échantillon de données portant sur la génomique humaine, du Centre de Recherche du Centre Hospitalier de l'Université de Montréal (CRCHUM), sera utilisé.

Faisant suite à un projet de fin d'études antérieurement réalisé par Mme Anna Klos, ce projet ira au-delà de ce que Mme Klos a accompli en utilisant plusieurs renseignements de

son rapport. Sommairement, le but ultime du projet est de fournir du matériel de calibre professionnel, qu'il s'agisse de la preuve de concept ou de la documentation y étant rattachée, pouvant être utilisé dans le cadre d'un cours à venir traitant du phénomène « big data ». L'environnement distribué créé doit donc pouvoir être reproduit sans heurts à la fin du projet grâce aux instructions fournies.

CHAPITRE 1

PROBLÉMATIQUE, CONTEXTE ET OBJECTIFS DU PROJET

Ce projet s'insère dans un contexte d'affaires où le Centre de Recherche du Centre Hospitalier de l'Université de Montréal (CRCHUM) est aux prises avec des problèmes avec un système d'identification de gènes et où l'ÉTS est désireuse d'amasser du matériel en vue d'un cours sur le « big data ».

Le CRCHUM, possédant bien au-delà de 150 millions d'enregistrements de données génomiques, utilise à l'heure actuelle un système permettant d'effectuer des recherches sur des gênes afin de, par exemple, trouver certaines variantes de gènes partageant des similarités. Selon les informations publiées sur le site Web du laboratoire Roueleau, « l'objectif principal du projet Synapse to Disease (de la synapse à la maladie ou S2D) est d'identifier des gènes causant ou prédisposant à des maladies du développement et du fonctionnement neuronal » (Laboratoire Guy Rouleau, 2012). S'appuyant sur une base de données relationnelle conventionnelle, le CRCHUM voit rapidement sa solution atteindre un plateau. En effet, plusieurs de leurs requêtes sont longues à effectuer et ont déjà demandé un remaniement de la base de données important. Leurs responsables voient donc, à l'horizon, un problème dans leur capacité de stocker et effectuer des requêtes efficaces sur les données. Le CRCHUM s'intéresse donc aux nouvelles technologies, comme HBase, qui pourraient assurer la pérennité de leur système d'informations génomiques en permettant d'ajouter sans cesse des nouvelles informations, sans pour autant saturer l'environnement ni ralentir les requêtes effectuées.

De son côté, le professeur Alain April réfléchit depuis un bon moment à créer de toutes pièces un cours portant sur le nouveau phénomène du « big data ». Lorsqu'il rencontra le Dr Rouleau, responsable du laboratoire Rouleau, lors d'une conférence, le professeur April vu immédiatement un potentiel dans le problème technologique de ce dernier.

En conséquence de ce qui précède, le projet aura des retombées sur les deux parties prenantes distinctes. D'un côté, le CRCHUM bénéficiera d'une preuve de concept qui peut servir comme prototype de solution à leurs problèmes de performance lors de requêtes sur plusieurs millions d'enregistrements. De l'autre côté, le professeur Alain April et son entourage pourront se servir des extrants du projet comme ressources pour monter un futur cours portant sur la technologie « big data ».

Le premier objectif à réaliser est de déterminer les facteurs variant dans les requêtes considérées problématiques du côté du CRCHUM. En effet, le projet de Mme Klos inclut des requêtes permettant peu de flexibilité, ce qui pourrait entraver l'utilité du projet pour le laboratoire Rouleau. Une fois les requêtes clairement définies, plusieurs alternatives, notamment MapReduce, seront considérées pour arriver à une solution.

Le second objectif consistera à installer et configurer de manière appropriée un environnement HDFS/HBase distribué sur les machines locales des développeurs. L'étudiante précédente, Mme Klos, avait réussi à installer HBase en mode « standalone », ce qui ne tirait pas profit du système de fichiers Hadoop. Pour des raisons de performance, HDFS sera utilisé et HBase s'en servira. Les environnements de développements utiliseront la version pseudodistribuée de HDFS, puisque seulement une machine est disponible pour propulser le système de fichiers. Ensuite, la grappe de trois serveurs fournis par l'ÉTS servira à installer et configurer HDFS/HBase en mode pleinement distribué.

Une fois l'environnement de développement local stable et fonctionnel, il faudra migrer les millions de données génomiques à partir de PostgreSQL vers HBase. Lorsque les giga-octets de données seront accessibles sur HBase en mode pseudodistribué, sur les machines des développeurs, il faudra exporter la table HBase dans la grappe de l'ÉTS. Ensuite, plusieurs tactiques de requêtes seront étudiées et développées de façon indépendante par chaque étudiant et, en fin de compte, la meilleure sera retenue pour servir dans la preuve de concept et la démonstration lors de la présentation orale finale. Avec les différentes façons d'effectuer des requêtes, des comparatifs de performances seront créés afin de comparer les

temps de requêtes entre la preuve de concept, la solution actuellement déployée au laboratoire Rouleau et la solution antérieurement proposée par Mme Klos.

Enfin, le dernier objectif du projet est de produire une application Web légère permettant de lancer des requêtes sur HBase avec des paramètres entrés par l'utilisateur. L'application n’offrira probablement pas beaucoup de fonctionnalité, mais permettra tout de même de paramétrer des requêtes et d’afficher leur résultat de manière plus ergonomique que de traiter directement avec le code source des requêtes.

CHAPITRE 2

SURVOL DES TECHNOLOGIES IMPLIQUÉES

2.1 Introduction à Hadoop

Tout d'abord, Hadoop est un framework libre, écrit en java, créé et distribué par la fondation Apache, destiné au traitement de données extrêmement volumineuses (de l'ordre des pétaoctets et plus) ainsi qu'à leur gestion intensive. Inspirée de plusieurs publications techniques rédigées par le géant Google, son objectif est de fournir un système de stockage et de traitement de données distribué, évolutif, extensible et constitué de matériel informatique de commodité, par exemple des postes informatiques conventionnels. La fondation Apache implémente à sa manière plusieurs des idées avancées par Google, notamment *HDFS* et *MapReduce*.

Comme le mentionne l'auteur Lars George dans son ouvrage HBase : The definitive guide, « Hadoop excelle dans le stockage de données de format arbitraire, semi-structuré ou même non structuré » (George, 2011). Par ailleurs, Hadoop est optimisé et pensé en fonction des systèmes dont la capacité de stockage de données est énorme et où les tâches sont souvent traitées en lot, étant donné les limitations au niveau du réseau ou de la charge des serveurs.

Utilisé dans un environnement distribué, Hadoop parallélise le traitement des données à travers de nombreux nœuds faisant partie d'une grappe d'ordinateurs, ce qui permet d'accélérer les calculs de grande envergure et de masquer la latence des opérations d'entrées et sorties par le biais d'une concurrence accrue. En outre, en plus d'accélérer le traitement d'opérations effectuées sur de très grandes données, Hadoop sait tirer profit de son système de fichiers distribué afin de répliquer certaines données vers des nœuds précis de sa grappe, rendant ainsi les données accessibles localement pour les machines devant les traiter.

Enfin, si un nœud de la grappe tombe en panne, Hadoop continue de fonctionner en redirigeant le travail aux autres machines de la grappe. Puisque, comme mentionné précédemment, le framework réplique par lui-même les données des nœuds au travers de la grappe, cette même grappe récupère bien souvent seule d'une machine en panne, tant au plan stockage que réalisation de tâches. Les administrateurs réseau n'ont donc qu'à repartir la machine qui est tombée et elle s'intègrera à nouveau dans la grappe automatiquement.

2.2 Hadoop File System (HDFS)

Hadoop File System (HDFS) est le système de fichiers utilisé par défaut par Hadoop. Tel que mentionné sur le site Web du projet, « HDFS crée des répliques multiples de blocs de données et les distribue sur les nœuds de calcul à travers une grappe pour permettre des calculs fiables et extrêmement rapides » (Apache Software Foundation, 2012). Puisqu'une image vaut mille mots, voici une représentation du système de fichiers d'Hadoop tirée du site Web de Cloudera :

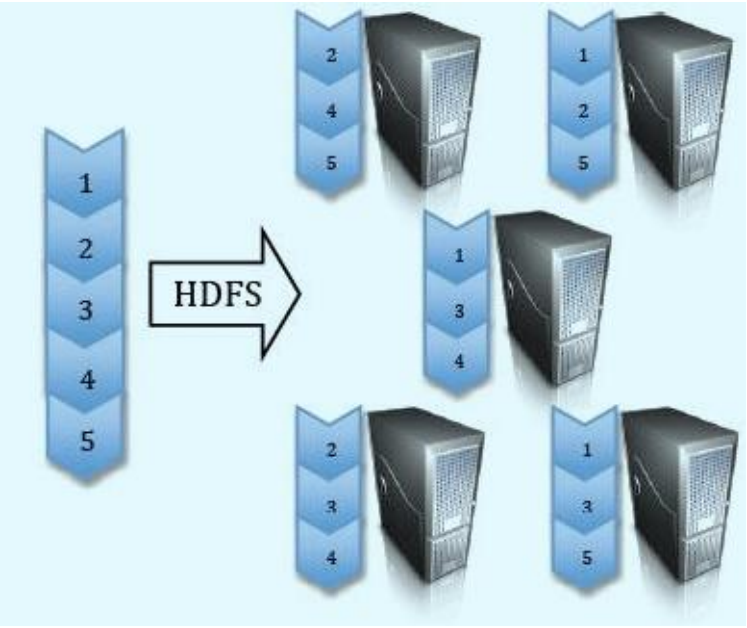


Figure 1 - Représentation visuelle de HDFS

L’image illustre HDFS séparant des fichiers en blocs de données redondants distribués sur les cinq machines du schéma.

2.2.1 Architecture HDFS

HDFS suit une architecture maître/esclave. En effet, une grappe HDFS est constituée d'un seul « NameNode », un serveur maître qui gère l'espace de noms du système de fichiers et réglemente l'accès aux fichiers par les clients. De plus, un certain nombre de « DataNodes », généralement un par noeud de la grappe, existent dans le but de gérer le stockage des données rattaché au noeud sur lequel ils fonctionnent. Tel que mentionné dans la documentation d'architecture d'Apache, « HDFS expose un espace de noms du système de fichiers et permet aux données de l'utilisateur d'être entreposées dans des fichiers » (Apache Software Foundation, 2012). À l'interne, un fichier est divisé en un ou plusieurs blocs et ces derniers sont stockés dans un ensemble de « DataNodes ». L'unique « NameNode » exécute les opérations sur le système de fichiers telles que l'ouverture, la fermeture et le renommage des fichiers et répertoires. Il détermine également sur quel « DataNode » chaque bloc de données sera stocké. Les « DataNodes » sont responsables du traitement des requêtes de lecture et d'écriture en provenance des clients du système de fichiers. En outre, ces derniers sont aussi responsables de la création, suppression et duplication des blocs de données, lorsqu'ordonnés par le serveur maître. Voici une image tirée de la documentation officielle d'Hadoop représentant les concepts de l'architecture HDFS :

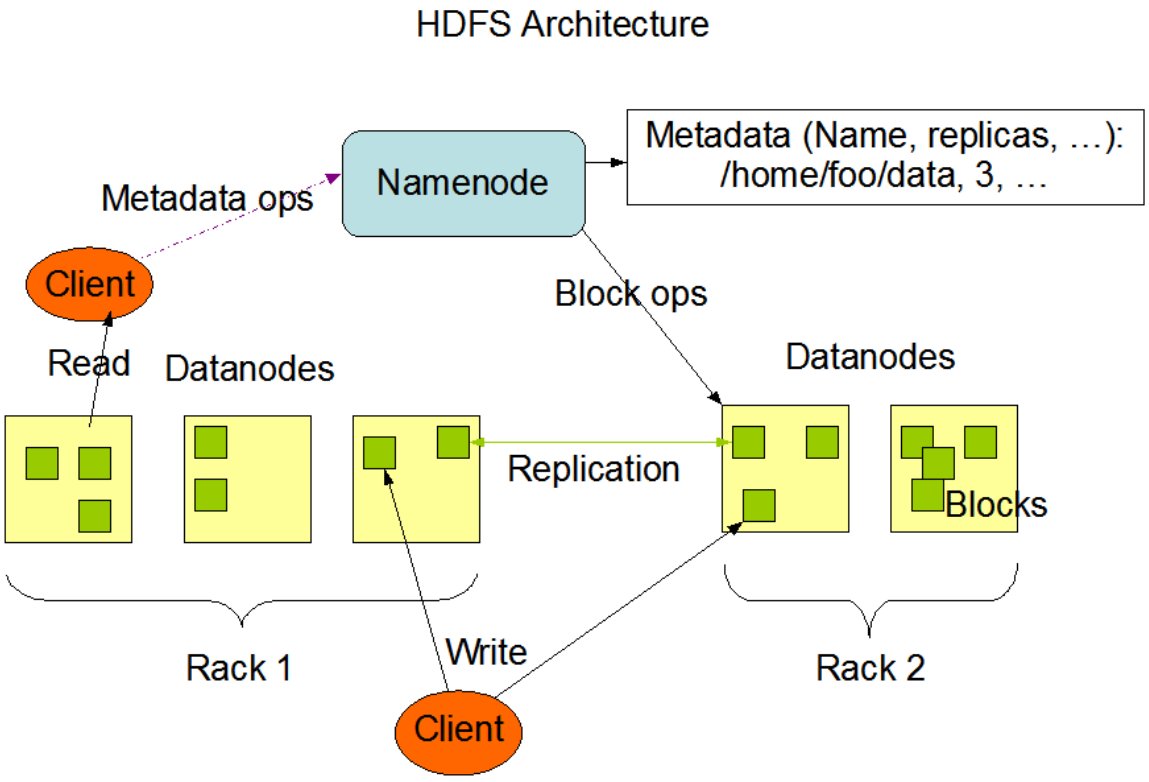


Figure 2 - Architecture HDFS

2.2.2 Réplication de données

Tel que mentionné par Apache dans leur documentation officielle, « HDFS est conçu pour stocker de manière fiable des fichiers très volumineux à travers des machines dans un très grand cluster » (Borthakur, 2009). HDFS stocke chaque fichier comme une séquence de blocs de données et tous les blocs d'un fichier, à l'exception du dernier, sont de la même taille. L'objectif visé dans la duplication des blocs à travers plusieurs fichiers est d'atteindre l'attribut de qualité de tolérance aux fautes. HDFS permet plusieurs configurations quant à la réplication des données, telles que la grosseur des blocs individuels ainsi que le nombre de répliques pour chaque fichier, ou encore le facteur de réplication par fichier individuel.

Le « NameNode », qui prend toutes les décisions concernant la réplication des blocs de données, reçoit périodiquement un pouls ("heartbeat") de chacun des serveurs esclaves.

Les esclaves peuvent envoyer au maître deux types de signal : « Heartbeat » et « Blockreport ». Le « Heartbeat » sert simplement à s'assurer que le serveur esclave est fonctionnel, alors que le « Blockreport » contient la liste de tous les blocs de données contenus sur le serveur expéditeur.

2.3 Hadoop MapReduce

Hadoop MapReduce est l'implémentation signée Apache du framework, introduit par Google, qui permet de facilement écrire des applications pouvant traiter des quantités massives de données, allant bien au-delà de plusieurs ensembles de téraoctets. Sa popularité en pleine croissance vient du fait qu'il effectue les requêtes et les opérations sur les données en parallèle sur des grappes de grande taille pouvant compter des milliers de noeuds construits avec du matériel de base à faible coût, ce qui le rend extrêmement performant et tolérant aux fautes en vertu du nombre de machines impliquées.

Le travail effectué par MapReduce se divise principalement en deux étapes: Map et Reduce. Tel que mentionné par Wikipédia, dans l'étape Map, « le nœud à qui est soumis un problème, le découpe en sous-problèmes, et les délègue à d'autres nœuds (qui peuvent en faire de même récursivement). Les sous-problèmes sont ensuite traités par les différents nœuds à l'aide de la fonction *Map* qui à un couple (clé, valeur) associe un ensemble de nouveaux couples (clé, valeur). Vient ensuite l'étape *Reduce*, où les nœuds les plus bas font remonter leurs résultats au nœud parent qui les avait sollicités. Celui-ci calcule un résultat partiel à l'aide de la fonction *Reduce* (réduction) qui associe toutes les valeurs correspondant à la même clé à une unique paire (clé, valeur). Puis il remonte l'information à son tour » (Wikipédia, 2012).

2.3.1 Map

Les maps sont des tâches individuelles qui transforment des enregistrements de données intrants en enregistrements de données intermédiaires. Il est important de noter qu'un enregistrement intermédiaire produit par une tâche Map n'est pas nécessairement de la même nature que son intrant. Par ailleurs, un enregistrement de données intrant peut produire un

nombre fort variable de paires clé-valeur intermédiaires, dépendamment des règles d'affaires de la tâche exécutée. Enfin, selon Wikipédia, « parce que chaque élément est calculé indépendamment et que la liste originale n'est pas modifiée, il est très facile de réaliser une opération *map* en parallèle » (Wikipédia, 2012). Bien entendu, la liste originale est ici synonyme de la liste contenant les enregistrements de données servant d'intrants aux tâches maps.

2.3.2 Combien de tâches Map?

Le nombre de tâches maps est généralement sélectionné en fonction de la taille totale des données à traiter. La documentation, fournie par Apache, de la version stable actuelle de Hadoop MapReduce (r1.0.3), suggère « entre 10 et 100 maps par noeud » (Apache, 2012). Cependant, leur documentation fait état que certaines configurations MapReduce ont déjà été jusqu'à 300 maps par noeuds pour certains traitements très légers. Enfin, il est suggéré que chaque map prenne au moins une minute à s'exécuter, étant donné que l'initialisation des tâches est assez coûteuse en temps.

2.4 HBase

À sa plus simple expression, la fondation Apache considère HBase comme « une base de données de type NoSQL ». En effet, HBase est la base de données du projet Hadoop et est en somme un entrepôt de données très volumineuses extensible et distribué.

2.4.1 NoSQL

NoSQL est un terme utilisé pour décrire une classe de systèmes de gestion de base de données qui se distinguent en n’adhérant pas au modèle conventionnel des systèmes de gestion de base de données relationnelle. En effet, une base de données NoSQL telle que HBase possède généralement, selon Wikipédia, les caractéristiques suivantes :

- « Elle n'utilise pas SQL comme moyen d'interroger les données, mais plutôt un autre, souvent plus simple, soit une interface de type API aux données.
- Elle peut ne pas garantir les propriétés transactionnelles ACID. De manière générale, seule la cohérence éventuelle est garantie. Ceci signifie que, compte tenu d'une période de temps suffisamment longue pendant laquelle aucun changement n'est envoyé à la base de données, il est convenable de s'attendre à ce que toutes les mises à jour de données soient propagées éventuellement à travers le système distribué.
- La base de données repose sur une architecture distribuée et tolérante aux fautes. » (Wikipédia, 2012)

2.4.2 Normalisation

À l'échelle, il est souvent de mise de conceptualiser le schéma de donnée différemment d'un simple schéma relationnel. Un bon terme pour décrire ce principe est « Denormalization, Duplication and Intelligent keys (DDI) ». Le principe DDI tourne autour de la notion de repenser à la manière dont les données dans un système de stockage dans le genre de Bigtable sont conservées, ainsi que de repenser aux approches d'interrogations de données dans le but d'utiliser efficacement cette nouvelle structure de données.

Une partie du principe repose sur la « dénormalisation » de données. « Dénormaliser » un schéma SQL implique, entre autres, de dupliquer les données dans plus d'une table afin que, lors d'une requête concernant ces données dupliquées, il n'y ait pas lieu d'effectuer une opération d'agrégation telle qu'un JOIN. Une autre option est de matérialiser les vues requises pour des requêtes à l'avance, dans le but d'optimiser les lectures de données en impliquant le moins possible de traitement supplémentaire.

HBase est une base de données qui requiert de repenser complètement l'organisation des données dans un schéma. En effet, HBase ne supporte pas le langage SQL, ni les fonctionnalités comme les JOIN. Les associations un à un, un à plusieurs et plusieurs à plusieurs doivent donc être converties vers une approche « dénormalisée ». Comme Lars

George le mentionne, « le support pour une conception soutenant de grandes tables clairsemées et orientées colonne élimine souvent la nécessité de normaliser les données et, dans le processus, les coûteuses opérations de jointure nécessaires pour agréger les données au moment de la requête » (George, 2011).

2.4.3 Architecture de HBase

HBase repose sur trois composants majeurs : la librairie client, un serveur maitre (« master server ») et plusieurs serveurs de région. Les serveurs de régions sont les éléments permettant une mise à l'échelle très vaste et peuvent être ajoutés ou retirés alors que le système est en marche. Le serveur maitre est responsable d'assigner des régions aux serveurs de régions et utilise Apache ZooKeeper pour faciliter cette tâche.

ZooKeeper est un projet libre séparé de Hadoop faisant aussi parti du groupe « Apache Software Foundation ». Tel que mentionné sur Wikipédia, « ZooKeeper est un logiciel de gestion de configuration pour systèmes distribués, basé sur le logiciel Chubby développé par Google » (Wikipédia, 2012). En effet, ZooKeeper est comparable au système Chubby utilisé par Google dans le cadre de Bigtable. ZooKeeper offre un accès semblable à celui d'un système de fichiers, ce qui inclut des fichiers et des répertoires, que les systèmes distribués peuvent utiliser pour négocier la propriété, enregistrer des services ou surveiller les mises à jour. Chacune des régions créer son nœud dans ZooKeeper, nœud que le serveur maitre emploie pour découvrir les serveurs esclaves disponibles. Ces nœuds éphémères sont aussi utilisés pour traquer différentes informations, par exemple des plantages de serveur.

Comme Lars George le mentionne dans son livre, « HBase utilise ZooKeeper pour s'assurer que seulement un serveur maitre fonctionne, pour stocker l'emplacement du "bootstrap" utilisé pour la découverte des régions, comme un registre pour les serveurs de région, ainsi que pour d'autres fins » (George, 2011).

Le serveur maitre, en plus de ce qui a été précédemment mentionné, est responsable de gérer la balance de charge des régions via les serveurs de région, de décharger les serveurs saturés et de déplacer des régions sur des serveurs moins chargés. Le serveur maitre ne fait pas partie en soit du stockage de données; il négocie et maintient l'état de la grappe, mais ne fournit aucun service, ce qui a comme conséquence qu'il est très peu chargé. Enfin, le maitre gère toute modification au schéma de données, comme la création de tables et de familles de colonnes. Voici, ci-dessous, une représentation graphique de l'architecture HBase :

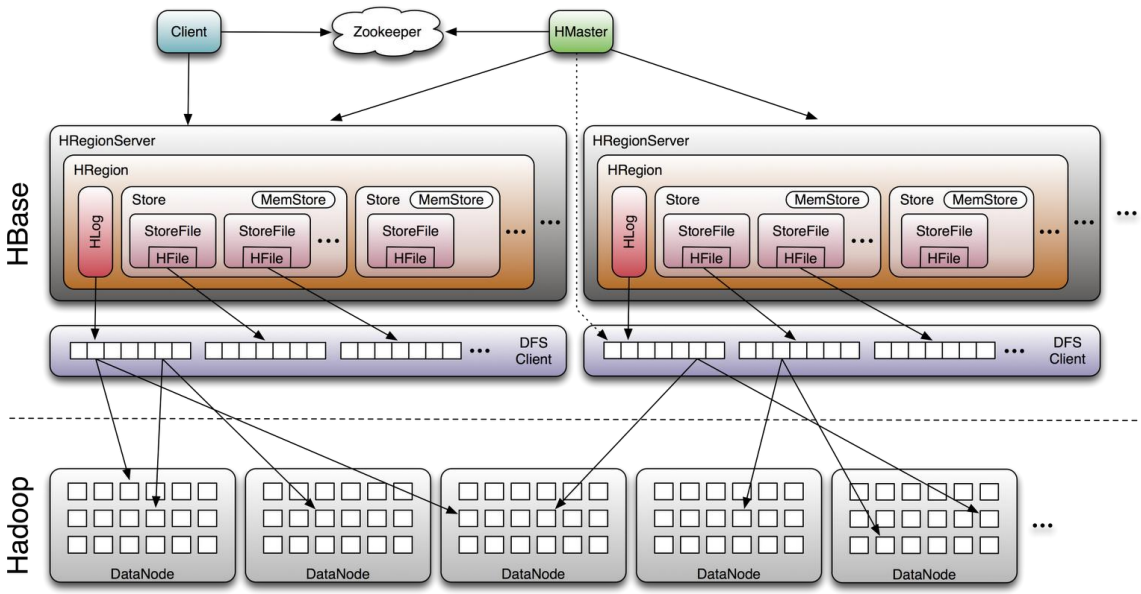


Figure 3 - Architecture HBase

2.4.4 Modes d'exécution

HBase possède deux modes dans lequel il peut s'exécuter : autonome (« standalone ») et distribué (« distributed »). L'installation par défaut de HBase est configurée pour fonctionner en mode autonome ou non distribué. Pour faire fonctionner HBase en mode distribué, il faut modifier certains fichiers de configuration dans le répertoire *conf* où est installé HBase.

En mode « Standalone », HBase ne fait pas usage du système de fichiers d'Hadoop. En effet, il utilise plutôt le système de fichiers localement installé sur la machine. Par exemple, sur une machine Linux, HBase utiliserait le système Ext4. De plus, le mode autonome fait aussi en sorte que tous les « daemons » liés à HBase, ainsi que ZooKeeper, s'exécutent dans une seule et même machine virtuelle Java (JVM). Bien que ce mode soit très facile à installer, il est très peu performant du fait qu'il ne tire aucunement profit de la force du nombre que fournit HDFS. Ce mode est donc conseillé pour les développeurs qui en sont à leurs premiers pas avec HBase et qui désirent comprendre et jouer avec le système. Il est strictement déconseillé d'utiliser ce mode dans un environnement de production puisqu'il ne permet aucune mise à l'échelle et ne fournit aucune réplication de données.

Le mode distribué est partagé en deux options : pseudodistribué (« pseudodistributed ») et pleinement distribué (« fully distributed »). L'unique différence entre les deux est qu'en mode pseudodistribué, tout s'exécute sur une seule machine, alors que le mode pleinement distribué requiert au minimum deux machines. Ainsi, en mode pseudodistribué, tous les « daemons » s'exécutent sur un seul nœud alors qu'en mode pleinement distribué, ils sont répartis sur plusieurs machines physiques de la grappe. Cependant, les deux modes distribués requièrent une instance du système de fichiers d'Hadoop (HDFS). Lars George suggère d'utiliser le mode pseudodistribué pour des fins de tests et de prototypages sur HBase. Cependant, en aucun cas ne faut-il utiliser ce mode ainsi que le mode autonome pour tester la performance de HBase. Tel que mentionné précédemment, les deux modes distribués nécessitent des configurations additionnelles au fichier *hbase-site.xml* situé dans le répertoire *conf* de l'installation HBase.

2.4.5 Concepts du modèle de données

Comme Anna Klos le mentionne dans son rapport de fin d'études, « la base de données HBase est composée de tables, de familles de colonnes, de colonnes, d'enregistrements, de cellules et de versions de colonnes » (Klos, 2012). Puisque HBase suit le principe de Bigtable, soit celui d'une base de données verticale, l'unité de base de HBase est la colonne.

Comme dans le modèle SQL conventionnel, une rangée est composée de plusieurs colonnes et est identifiée par une clé unique. Ainsi, les tables sont construites avec une combinaison de colonnes et rangées. C'est ici que s'arrêtent les ressemblances entre le modèle relationnel et le modèle orienté colonnes de HBase.

HBase ajoute plusieurs concepts à ce qui a été explicité au dernier paragraphe. En effet, la base de données ajoute une dimension de version au niveau des colonnes. Ceci signifie qu'une colonne donnée peut avoir plusieurs versions, avec chacune une valeur distincte contenue à l'intérieur d'une cellule isolée des autres. Voici une image pouvant aider à visualiser le concept de cette dimension ajoutée :

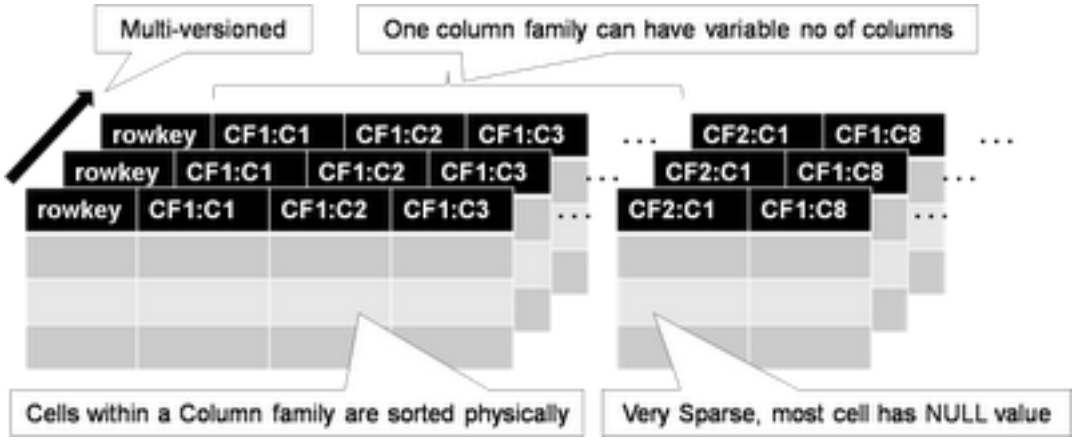


Figure 4 - Dimension des versions dans HBase

Un autre ajout au modèle relationnel est l'ordonnancement lexicographique des clés de rangées. Cet ordonnancement présent en tout temps agit sensiblement comme un index sur une clé primaire dans le cadre d'un modèle de données relationnel. Comme le mentionne Lars George dans son ouvrage HBase : The Definitive Guide, « dans le tri lexicographique, chaque clé est est comparée au niveau binaire, octet par octet, de gauche à droite » (George, 2011). Voici un exemple de tri lexicographique effectué dans HBase :

Tableau 1 - Exemple de tri alphanumérique

ROW	COLUMN+CELL
row-1	column=cf1:, timestamp=129707332 …
row-10	column=cf1:, timestamp=129707344 …
row-11	column=cf1:, timestamp=129707355 …
row-2	column=cf1:, timestamp=129707366 …
row-22	column=cf1:, timestamp=129707385 …

Enfin, un dernier concept est ajouté à HBase : les familles de colonnes. Les familles de colonnes servent à regrouper des colonnes et à tracer des séparations sémantiques entre elles. De plus, au niveau physique, les familles de colonnes sont sauvegardées ensemble dans un fichier de type HFile. Lorsque les familles de colonnes sont bien définies, la recherche de données est moins gourmande puisque HBase n'a pas besoin de regarder dans tous les fichiers HFile et retourner l'information si elle n'est pas nécessaire à la requête. Les familles de colonnes ne sont pas aussi dynamiques que les colonnes. Elles doivent être définies lors de la création de la table et ne doivent pratiquement pas changer en cours de route. À l'opposé, les colonnes peuvent être instanciées et ajoutées aux tables à tout moment. Comme le mentionne la documentation officielle de HBase, « toutes les colonnes membres d'une famille de colonnes ont le même préfixe » (Apache Software Foundation, 2012). Le caractère « : » délimite la famille de colonne du qualificatif de la colonne. Par exemple, *logti :log120* et *logti :log430* sont deux colonnes membres de la famille de colonnes *logti* représentant des cours donnés à l'ÉTS. Une note importante sur le nombre de familles de colonnes est présente dans la documentation officielle du projet de la fondation Apache : « HBase n'est actuellement pas à l'aise avec quoi que ce soit au-dessus de deux ou trois familles de colonnes dans un schéma » (Apache Software Foundation, 2012). Allant dans le sens contraire de cette limitation au niveau des colonnes elles-mêmes, HBase supporte des milliers, voir millions de colonnes par familles de colonnes (et donc par table).

CHAPITRE 3

ENVIRONNEMENT DE DÉVELOPPEMENT MAC OS X LION

3.1 Installation de Hadoop en mode pseudo-distribué

Tout d'abord, voici la configuration du poste de travail OS X :

Figure 5 - Configuration du poste de travail

Bien que Mac OS X ne soit pas officiellement recommandé par Apache, le présent travail aura tôt prouvé que Hadoop et HBase sont très stables sur ce dernier. La machine, en soit, est un PC construit sur mesure sur lequel Mac OS X Lion a été installé. OS X réside sur un disque dur Solid State (SSD) de 120 gigaoctet. Le reste des spécifications de la machine est indiqué sur l'image.

Plutôt que d'exécuter HDFS sous le compte « root », l'installation décrite dans ce chapitre installe et configure HDFS pour qu'il s'exécute avec le compte courant de l'utilisateur.

Avant de débuter l'installation de Hadoop, il est important que le poste accueillant HDFS ait un accès SSH. En effet, Hadoop doit configurer plusieurs rôles de serveur lorsqu'il démarre et cette opération est exécutée via SSH. Ainsi, Hadoop regardera à l'adresse localhost pour configurer les divers rôles. Dans le but d'éviter d'entrer le mot de passe SSH à chaque démarrage et arrêt de HDFS, il est recommandé de s'ajouter soi-même à la liste de clés autorisées sur SSH. La commande bien connue « ssh-copy-id » effectue tout le travail, mais ne vient malheureusement pas par défaut avec la distribution OS X. Pour palier à ce manque, l'option la plus simple est d'utiliser le gestionnaire d'applications Homebrew pour télécharger l'utilitaire ssh-copy-id. Il est très facile d'installer Homebrew en suivant les indications fournies sur leur site Web officiel à l'adresse suivante : http://mxcl.github.com/homebrew/. Une fois Homebew installé, il suffit de lancer la commande suivante dans la console :

```
$ brew install ssh-copy-id
```

Enfin, voici la commande pour ajouter le poste local à la liste des clés SSH autorisées à se connecter sans mot de passe :

```
$ ssh-copy-id USAGER@localhost
```

Pour vérifier que SSH fonctionne adéquatement, rien de tel que de se connecter sans mot de passe :

```
$ ssh localhost
```

3.2 Installation de Hadoop

La plus récente version stable de Hadoop est disponible sur l'un des multiples miroirs affichés sur le site d'Apache suivant : http://www.apache.org/dyn/closer.cgi/hadoop/common/. Les archives peuvent être téléchargées d'une multitude de façons, mais la façon utilisée dans ce projet est la suivante utilise la commande **wget**. Une fois de plus, l'utilitaire **wget** n'est pas livré avec Mac OS X Lion et Homebrew possède la solution facile pour l'installer :

```
$ brew install wget
```

La dernière version stable de Hadoop peut être téléchargée à partir du site http://apache.mirror.rafal.ca/hadoop/common/stable/, un des miroirs officiellement publiés par Apache. Au moment d'écrire ces lignes, la version stable la plus à jour est 1.0.3. Voici comment télécharger Hadoop :

```
$ cd
$ mkdir tmp
$ cd tmp
$  wget  http://apache.mirror.rafal.ca/hadoop/common/stable/hadoop-
1.0.3.tar.gz
$  wget  http://apache.mirror.rafal.ca/hadoop/common/stable/hadoop-
1.0.3.tar.gz.mds
```

Afin de vérifier l'intégrité de l'archive téléchargée, la signature MD5 de l'archive est lue et comparée au référentiel MD5 Sum « .mds » :

```
$ md5 *.gz
$ cat hadoop-1.0.3.tar.gz.mdas
```

En exécutant cette simple et rapide vérification, plusieurs maux de tête sont évités en certifiant que l'archive n'est pas corrompue ou malformée. Vient ensuite l'extraction de l'archive dans le répertoire $HOME et la création d'un lien symbolique qui pourra simplifier l'accès à l'installation de Hadoop :

```
$ cd
$ tar –zxvfp tmp/*.tar.gz
$ ln –s hadoop-* hadoop
```

Afin que Mac OS X puisse détecter et lancer les fichiers binaires de Hadoop à partir de n'importe quel répertoire, il faut ajouter le répertoire « bin » de l'installation Hadoop dans le fichier « .bash_profile », lui-même situé dans le répertoire $HOME en ajoutant cette ligne :

```
export PATH="$HOME/hadoop/bin:$PATH"
```

Enfin, la commande suivante permet de prendre en ligne de compte la ligne nouvellement ajoutée au « .bash_profile » sans nécessiter de fermer et rouvrir la session utilisateur :

```
$ source ~./bash_profile
```

Hadoop est maintenant supposé être correctement extrait et déployé sur la machine. La suite consiste à configurer HDFS en termes de port, mémoire allouée, etc.

3.2.1 Configuration de Hadoop File System

Le fichier *hadoop-env.sh* renferme toutes les configurations quant à l'environnement Java utilisé par Hadoop. La variable **JAVA_HOME** dans le fichier doit pointer vers le répertoire racine de Java. Sous Mac OS, le programme java_home faisant partie de la librairie « libexec » permet de déterminer facilement et sur demande l'emplacement de l'installation

Java sur la machine. De cette façon, l'emplacement du répertoire racine de l'installation Java peut être écrit comme suit dans le fichier *hadoop-env.sh* :

```
# Fichier ~/hadoop/conf/hadoop-env.sh
export JAVA_HOME=$(/usr/libexec/java_home)
```

Comme le mentionne Apache dans sa documentation sur la configuration des grappes Hadoop, à tout le moins, « vous devez spécifier le répertoire JAVA_HOME de sorte qu'il est correctement configuré sur chaque nœud à distance » (Apache Software Foundation, 2012).

Outre la configuration obligatoire JAVA_HOME, il existe d'autres options méritant d'être considérées. Par souci de concision, une seule autre est présentée dans ce rapport, mais la liste complète des options peut être consultée à l'adresse suivante : http://hadoop.apache.org/common/docs/r1.0.3/cluster_setup.html. La configuration, par défaut, pour la taille de la pile d'exécution (« heap size ») est de 1000 mégaoctets. La machine sur laquelle cet environnement de développement a été installé possédant 6 gigaoctets de mémoire vive, la taille maximale de la pile d'exécution de Hadoop a été relevée à 2000 mégaoctets grâce à la configuration suivante :

```
# Fichier ~/hadoop/conf/hadoop-env.sh
export HADOOP_HEAPSIZE=2000
```

Une fois ces paramètres critiques ajustés dans *hadoop-env.sh*, des configurations additionnelles à propos du répertoire de données local sont offertes dans le fichier *core-site.xml*. Par définition, ce fichier XML contient les configurations spécifiques à chaque site de la grappe. Puisque Hadoop fonctionne sous le compte utilisateur courant, il est pertinent de mettre le répertoire de données de HDFS dans le dossier HOME, mais n'importe quel dossier pour lequel l'utilisateur courant possède les droits d'accès en lecture et écriture fera.Voici les configurations utilisées pour l'environnement sous Mac OS X :

```
<configuration>

<property>
  <name>hadoop.tmp.dir</name>
  <value>/Users/YOUR_ACCOUNT/.hdfs.tmp</value>
  <description>A base for other temporary
directories.</description>
</property>

<property>
        <name>fs.default.name</name>
        <value>hdfs://localhost:8020</value>
</property>

</configuration>
```

Par ailleurs, Hadoop offre certains réglages additionnels dans le fichier de configuration *hadoop-site.xml*. Une fois de plus, la documentation officielle est la meilleure référence pour connaître tout ce qui est offert. Dans le cadre de ce projet, seule l'option de réplication a été utilisée afin de spécifier de conserver uniquement une copie des données, puisque la réplication des données n'accélère en rien la vitesse de lecture et d'écriture en mode pseudodistribué. Les lignes suivantes du fichier *hadoop-site.xml* expriment ce réglage :

```
<configuration>

    <property>
        <name>dfs.replication</name>
        <value>1</value>
    </property>

</configuration>
```

Enfin, le dernier fichier de configuration touché par l'environnement OS X utilisé dans le cadre du projet est *mapred-site.xml*, un fichier de réglages destiné à l'engin MapReduce. Ce fichier permet de régler des options telles que l'emplacement du « job tracker », le nombre

maximum et minimum de « jobs » MapReduce concurrentes, etc. Voici la configuration utilisée pour le projet, inspirée du blogue Jianwen Wei situé à l'adresse http://jianwen.me :

```
<configuration>

    <property>
        <name>mapred.job.tracker</name>
        <value>localhost:8021</value>
    </property>

    <property>
        <name>mapred.tasktracker.map.tasks.maximum</name>
        <value>4</value>
    </property>

    <property>
        <name>mapred.tasktracker.reduce.tasks.maximum</name>
        <value>4</value>
    </property>

</configuration>
```

Sur Mac OS X Lion, une erreur peu intuitive survient lors du démarrage de Hadoop. L'erreur s'illustre par le message « *Unable to load realm info from SCDynamicStore* » lors de l'initialisation du nœud HDFS. Le problème est connu, documenté et suivi par la communauté. Il est possible de consulter le problème ainsi que son correctif à l'adresse suivante : https://issues.apache.org/jira/browse/HADOOP-7489. Pour corriger ce bogue, il suffit d'ajouter cette dernière configuration au fichier *hadoop-env.sh* :

```
# File: ~/hadoop/conf/hadoop-env.sh
export HADOOP_OPTS="-Djava.security.krb5.realm=OX.AC.UK -
Djava.security.krb5.kdc=kdc0.ox.ac.uk:kdc1.ox.ac.uk"
```

À ce point, tous les réglages nécessaires au bon fonctionnement de Hadoop dans l'environnement distribué sont entrés. Lors de la première mise en marche, il faut initialiser le système de fichiers en utilisant la commande suivante :

```
$ hadoop namenode -format
```

Une fois ceci complété, il est possible de démarrer Hadoop à l'aide de la commande suivante :

```
$ start-all.sh
```

La commande **jps**, qui dresse une liste des JVMs en fonction dans le système, permet de vérifier que tous les processus Hadoop sont en marche :

```
$ jps
30920 TaskTracker
30723 JobTracker
30944 Jps
30256 NameNode
30654 SecondaryNameNode
30456 DataNode
```

3.3 Installation de HBase dans l'environnement

3.3.1 Installation de HBase

Un peu comme pour l'installation de Hadoop, l'installation de HBase comprend les phases installation et configuration. Pour débuter, la plus récente version stable de HBase peut être téléchargée à partir de l'un des miroirs officiels d'Apache. Au moment d'écrire ces lignes, la version stable est 0.92.1. Comme pour Hadoop, le miroir choisit fut celui de rafal.ca, à l'adresse http://apache.mirror.rafal.ca/hbase/stable/. Pour télécharger HBase, il suffit d'entrer les commandes suivantes en console :

```
$ cd

$ cd tmp

$ wget http://apache.mirror.rafal.ca/hbase/stable/hbase-
0.92.1.tar.gz
```

Comme pour Hadoop, il est très important de vérifier l'intégrité de l'archive téléchargée en vérifiant sa somme MD5. Ceci est possible avec la commande suivante :

```
$ md5 *.gz
$ cat hbase-0.92.1.tar.gz.mds
```

Une fois de plus, comme pour l'installation de Hadoop, cette procédure installe HBase dans le répertoire $HOME de l'utilisateur courant. Un lien symbolique permet aussi d'accéder plus facilement au répertoire d'installation de HBase, en plus de permettre de changer la destination du lien symbolique dans le futur, par exemple dans le cas d'une mise à jour de HBase, sans pour autant changer les configurations de toutes les applications y faisant référence. Ces opérations sont accomplies à l'aide des commandes suivantes :

```
$ cd
$ tar -xzvpf tmp/hbase-0.92.1.tar.gz
$ ln -s hbase-0.92.1 hbase
```

Afin que Mac OS X puisse détecter et lancer les fichiers binaires de HBase à partir de n'importe quel répertoire, il faut ajouter le répertoire « bin » de l'installation HBase dans le fichier « .bash_profile » situé dans le répertoire $HOME en ajoutant cette ligne :

```
export PATH="$HOME/hbase/bin:$PATH"
```

Enfin, la commande suivante permet de mettre à jour la variable $PATH, sans nécessiter de fermer et rouvrir la session utilisateur :

```
$ source ~./bash_profile
```

HBase est maintenant supposé être correctement extrait et déployé sur la machine. La suite consiste à configurer HBase afin de lui indiquer l'emplacement de l'installation Java ainsi que le port sous lequel HBase pourra s'exécuter.

3.3.2 Configuration de HBase

Les configurations de Hadoop et HBase partagent un élément essentiel : l'emplacement du répertoire JAVA_HOME. Exactement comme pour l'installation du système de fichiers, l'emplacement du répertoire est indiqué dans le fichier « hbase-env.sh » (similarité « hadoop-env.sh » et « hbase-env.sh ») de la façon suivante :

```
# Fichier ~/hbase/conf/hbase-env.sh

export JAVA_HOME=$(/usr/libexec/java_home)
```

De plus, le bogue décrit dans la section sur la configuration de Hadoop affecte aussi HBase. Le problème, identifié par le numéro Hadoop-7489, affecte uniquement les systèmes Mac OS

X. Celui-ci est corrigé avec la ligne suivante à ajouter au fichier de configuration « hbase-env.sh » :

```
# Fichier ~/hbase/conf/hbase-env.sh

export HBASE_OPTS="-XX:+UseConcMarkSweepGC -
Djava.security.krb5.realm=OX.AC.UK -
Djava.security.krb5.kdc=kdc0.ox.ac.uk:kdc1.ox.ac.uk"
```

Puisque l'environnement de développement est configuré pour utiliser HDFS en mode pseudodistribué, il faut l'indiquer à HBase puisque, par défaut, la base de données est paramétrée pour utiliser le système de fichiers local. Pour ce faire, il suffit d'inscrire ces lignes dans le fichier « hbase-site.xml » situé dans le répertoire *~/hbase/conf/hbase-site.xml* :

```
# Fichier ~/hbase/conf/hbase-site.xml

<configuration>
        <property>
            <name>hbase.rootdir</name>
            <value>hdfs://localhost:9000/hbase</value>
            <description>Check  hadoop/conf/core-site.xml  in  hadoop
            for NameNode's port number.</description>
        </property>
</configuration>
```

HBase, en mode pseudodistribué, nécessite que HDFS soit en fonction pour démarrer. Il faut donc, dans un premier temps, démarrer HDFS et ensuite HBase avec les commandes suivantes :

```
$ start-all.sh
```

Enfin, comme pour l'installation de HDFS, la commande **jps** permet de visualiser les JVMs en fonction et ce coup-ci, la liste inclut HBase :

```
$ jps

16559 Jps
16337 TaskTracker
16464 HMaster
15960 NameNode
16237 JobTracker
```

3.4 Installation de PostgreSQL

3.4.1 Téléchargement des fichiers et installation

Les données du CRCHUM sont actuellement stockées dans une base de données PostgreSQL. Afin d'y accéder pour les migrer vers HBase, il faut installer un serveur Postgres. Puisque HomeBrew a été installé auparavant, l'installation de Postgres est grandement facilitée. Il suffit d'utiliser HomeBrew et d'inscrire la commande suivante :

```
$ brew install postgresql
```

HomeBrew se charge d'installer toutes les librairies nécessaires au fonctionnement de Postgres. Cependant, l'erreur suivante survient

```
PGError (could not connect to server: Permission denied    Is    the
server running locally and accepting     connections on Unix domain
socket "/var/pgsql_socket/.s.PGSQL.5432"? ):
```

En effectuant quelques recherches, il apparaît que Mac OS X Lion est livré par défaut avec un serveur Postgres caché, non accessible directement et qui est utilisé par certains processus internes critiques de Mac OS. Comme le mentionne Tammer Saleh sur son site Web, il s'avère que les deux serveurs « PostgreSQL se piétinent les uns les autres » (Saleh). Puisqu'il est impossible de désinstaller et supprimer le serveur livré avec OS X, il faut tout simplement

indiquer, pour l'utilisateur, quel serveur utiliser avant l'autre. En consultant le fichier « .basrc », voici comment la variable d'environnement $PATH était configurée :

```
PATH=$PATH:/usr/local/Cellar/postgresql/9.1.4/bin
```

Ceci signifiait que le serveur Postgres livré avec OS X avait préséance sur celui qui venait d'être ajouté au système. Ceci étant incorrect, il faut inverser cette configuration en modifiant la ligne pour la suivante :

```
PATH=/usr/local/Cellar/postgresql/9.1.4/bin:$PATH
```

Comme précédemment, il faut exécuter la commande suivante pour que les nouveaux paramètres de « .bashrc » soient pris en compte sans fermer et rouvrir la session utilisateur :

```
$ cd
$ source .bashrc
```

3.4.2 Création et configuration de l'utilisateur postgres

Comme le mentionne la documentation officielle de PostgreSQL, « comme avec n'importe quel démon serveur qui est accessible au monde extérieur, il est conseillé de lancer PostgreSQL sous un compte utilisateur séparé. Ce compte d'utilisateur ne doit posséder des données qui sont gérées par le serveur, et ne doit pas être partagé avec d'autres démons. (Par exemple, utiliser l'utilisateur nobody est une mauvaise idée) » (PostgreSQL, 2010). Généralement, pour ajouter un utilisateur dédié sur un système Unix, la commande **useradd** ou **adduser** est disponible. Cependant, puisque Mac OS X utilise Open Directory pour gérer ses comptes utilisateurs, il n'y a aucune commande **useradd** à proprement dit. Au lieu de cela, un utilitaire de services de répertoire pour ajouter un nouvel utilisateur est disponible. Cet utilitaire varie en fonction de la version du système d'exploitation Mac et, dans le cas de

Lion, l'utilitaire est **dscl**, disponible en ligne de commande. Il est donc question de manuellement créer un compte utilisateur dédié au serveur Postgres et de lui assigner son propre groupe aussi, de sorte que tout fichier lié à la base de données et possédant des droits d'accès en écriture pour le groupe ne soit pas modifié par un tiers utilisateur.

Tout d'abord, il faut trouver des identifiants (ID) d'utilisateur et de groupe qui sont disponibles, soit non utilisés par d'autres processus. Pour ce faire, les commandes à utiliser pour visualiser les listes d'utilisateurs et de groupes sont les suivantes :

```
$ sudo dscl . -list /Groups PrimaryGroupID
$ sudo dscl . -list /Users UniqueID
```

Ici, il sera assumé que l'identifiant de groupe 229 et l'identifiant d'utilisateur 100 sont disponibles sur l'environnement. Il faut tout d'abord créer le groupe auquel l'utilisateur dédié sera assigné. En ligne de commande, ceci est effectué en entrant ces commandes :

```
$ sudo dscl . -create /Groups/_postgres

$ sudo dscl . -create /Groups/_postgres PrimaryGroupID 229
```

La raison pour laquelle un trait de soulignement devant le groupe « postgres » est que, depuis Mac OS X 10.6, les noms de processus (« daemon ») sont précédés d'un trait de soulignement. Cependant, nul besoin de se souvenir du trait, puisque la commande créer aussi un alias sans trait de soulignement, soit « postgres ». Pour créer le compte utilisateur, il suffit d'entrer les commandes suivantes :

```
$ sudo dscl . -create /Users/_postgres

$ sudo dscl . -create /Users/_postgres UniqueID 100

$ sudo dscl . -create /Users/_postgres PrimaryGroupID 229

$ sudo dscl . -create /Users/_postgres UserShell /bin/bash

$ sudo dscl . -create /Users/_postgres RealName "PostgreSQL Server"
```

L'utilisateur est maintenant créé. Son répertoire $HOME est « /usr/local/pgsql », de sorte que tous les fichiers de données créés par PostgreSQL lui appartiennent. Le compte ne possède aucun mot de passe et ceci est intentionnel, puisque cela empêche toute personne sauf l'utilisateur « root » de s'y connecter. Pour utiliser le compte postgres, il faut exécuter la commande suivante en ligne de commande :

```
$ sudo su - postgres
```

3.4.3 Initialisation de la base de données

Puisque le compte utilisateur a été configuré pour utiliser le répertoire « /usr/local/pgsql », le répertoire de données du serveur PostgreSQL sera ce dernier. Cependant, à cause que le compte utilisateur « postgres » ne possède aucun privilège, il faut créer le répertoire « /usr/local/pgsql » avec l'utilisateur courant et faire en sorte que l'utilisateur « postgres » en soit le propriétaire :

```
$ sudo mkdir /usr/local/pgsql

$ sudo chown -R _postgres:_postgres /usr/local/pgsql
```

Vient alors l'initialisation de la base de données :

```
$ initdb -D /usr/local/pgsql/data
```

Une fois le serveur PostgreSQL initialisé, il faut créer une base de données. Dans l'environnement de développé traité ici, la base de données fut nommée « Postgres », mais n'importe quel nom fera. La commande pour créer la base de données est la suivante :

```
$ createdb postgres
```

La prochaine étape est de mettre en marche la base de données afin qu'elle soit accessible. Il existe bien entendu une commande pour effectuer cette tâche, mais un outil graphique simple et efficace existe pour démarrer et stopper le serveur Postgres. En effet, John T Wang a eu l'idée de développer cette application qui s'installe dans le panneau des préférences système de Mac OS X après qu'il ait eu une expérience frustrante avec les divers paramètres de connexion dans la ligne de commande. L'application peut être téléchargée sur GitHub à l'endroit qui suit : https://github.com/jwang/pgpane. Voici un aperçu de la fenêtre principale de l'application ainsi que de la fenêtre des paramètres de connexion:

Figure 6 - Fenêtre principale de l'utilitaire PostgreSQL

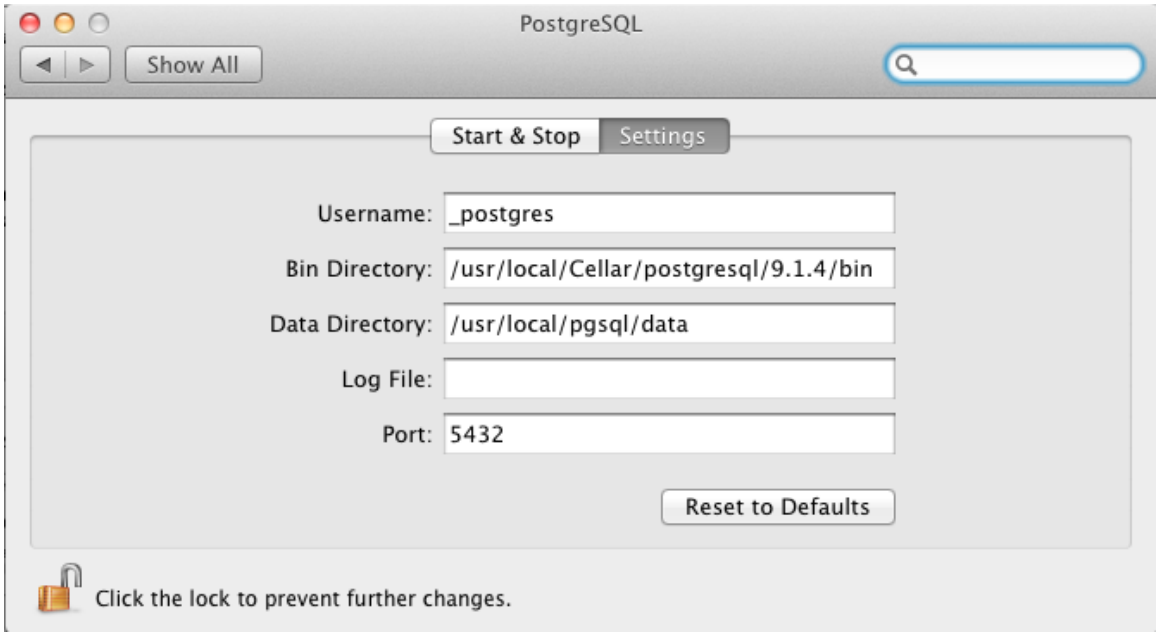


Figure 7 - Fenêtre des paramètres de connexion de l'utilitaire PostgreSQL

Une fois les paramètres entrés, le démarrage du serveur devrait se faire sans heurt et le statut devrait indiquer, en vert comme dans l'image, le texte « Running ». La base de données

PostgreSQL qui servira de source de données à HBase est dorénavant installée et opérationnelle avec son propre utilisateur dédié.

3.5 Installation de l'utilitaire Sqoop

Comme le décrit Apache, Sqoop « est un outil conçu pour transférer efficacement les données en vrac entre Apache Hadoop et des bases de données plus conventionnelles telles que les bases de données relationnelles » (Apache Software Foundation, 2012). Sqoop sera l'outil qui va permettre une migration du modèle de données relationnel sur PostgreSQL vers le modèle NoSQL sur HBase. Cette migration se fera presque automatiquement et simplifiera de manière significative l'opération. Pour installer Sqoop sous Mac OS X, il faut procéder manuellement. Dans un système Linux, le gestionnaire d'applications Aptitude permet une installation automatique. Cependant, HomeBrew n'étant pas supporté par Apple, l'utilitaire est plus limité qu'Aptitude. Il faut donc télécharger l'archive de Sqoop à partir d'un des miroirs offerts par Apache à la page suivante : http://www.apache.org/dyn/closer.cgi/sqoop/. La version à installer dans l'environnement de développement est au choix (1.4.0 ou 1.4.1 sont offertes), mais celle utilisée et testée pendant le projet est 1.4.1.

Une fois l'archive en main, il faut l'extraire à l'endroit de son choix. Certains veulent extraire toutes les applications dans le répertoire /usr/local/lib, mais ici, le répertoire de destination est $HOME, comme pour les installations de Hadoop et HBase. L'extraction s'effectue avec la commande suivante :

```
$ cd

$ tar -zxvfp tmp/sqoop-1.4.1-incubating__hadoop-0.23.tar.gz
```

Afin de pouvoir lancer la commande **sqoop** de partout en ligne de commande, il faut ajouter le répertoire des fichiers binaire de Sqoop à la variable $PATH. Ceci est accompli en ajoutant la ligne suivante au fichier « .bash_profile » :

```
export PATH="$HOME/sqoop/bin:$PATH"
export HADOOP_HOME="$HOME/hadoop"
```

La seconde ligne, qui définit la variable $HADOOP_HOME, est requise par Sqoop. Ironiquement, HBase lance un message d'avertissement à chaque démarrage qui indique que l'usage de $HADOOP_HOME est déprécié, mais il semble que Sqoop ne puisse fonctionner sans cela. Enfin, pour activer les changements sur le champ :

```
$ cd
$ source .bash_profile
```

Sqoop est dorénavant censé être installé et accessible de partout. Cependant, il reste une dernière étape pour que le tout soit fonctionnel. Afin de transférer les données de PostgreSQL à HBase, l'utilitaire Sqoop doit se connecter à la base de données PostgreSQL et requiert donc un pilote JDBC. Puisque la JVM courante sur le poste est 1.6.X, il faut télécharger le pilote JDBC 4. Le pilote est disponible pour télécharger à l'adresse suivante : http://jdbc.postgresql.org/download.html. Une fois le pilote téléchargé, il faut placer le fichier « postgresql-9.1-902.jdbc4.jar » dans la racine du répertoire d'installation de Sqoop. L'installation et la configuration de Sqoop pour lui permettre de se connecter à la base de données PostgreSQL sont maintenant complétées.

CHAPITRE 4

MIGRATION DES DONNÉES VERS HBASE

4.1 Schéma PostgreSQL

Tel que discuté avec le Professeur Alain April au début du projet, tout le schéma PostgreSQL n'a pas nécessairement besoin d'être migré vers HBase. L'idée est donc d'obtenir une solution hybride où les tables de petite taille ainsi que les tables ne faisant pas partie des requêtes posant problème ne seront pas transposées dans HBase. Seules les tables très lourdes (sujettes à saturation) et celles incluses dans les grosses requêtes seront migrées. Bien entendu, cette solution peut varier au fil du temps si le besoin s'en fait sentir.

Reprenant les travaux d'Anna Klos, le schéma composé de treize tables peut être, comme elle le mentionne dans son propre rapport, « regroupé en trois sous-schémas » (Klos, 2012). Le premier sous-schéma comprend les tables de références pour décrire les algorithmes de séquençage ainsi que leurs relations avec les tables de « pipeline » et de dépistage. Comme ces tables sont peu volumineuses, elles ne sont pas candidates à une migration vers HBase dans le cadre de ce projet d'étude. Voici, ci-dessous, le premier sous-schéma :

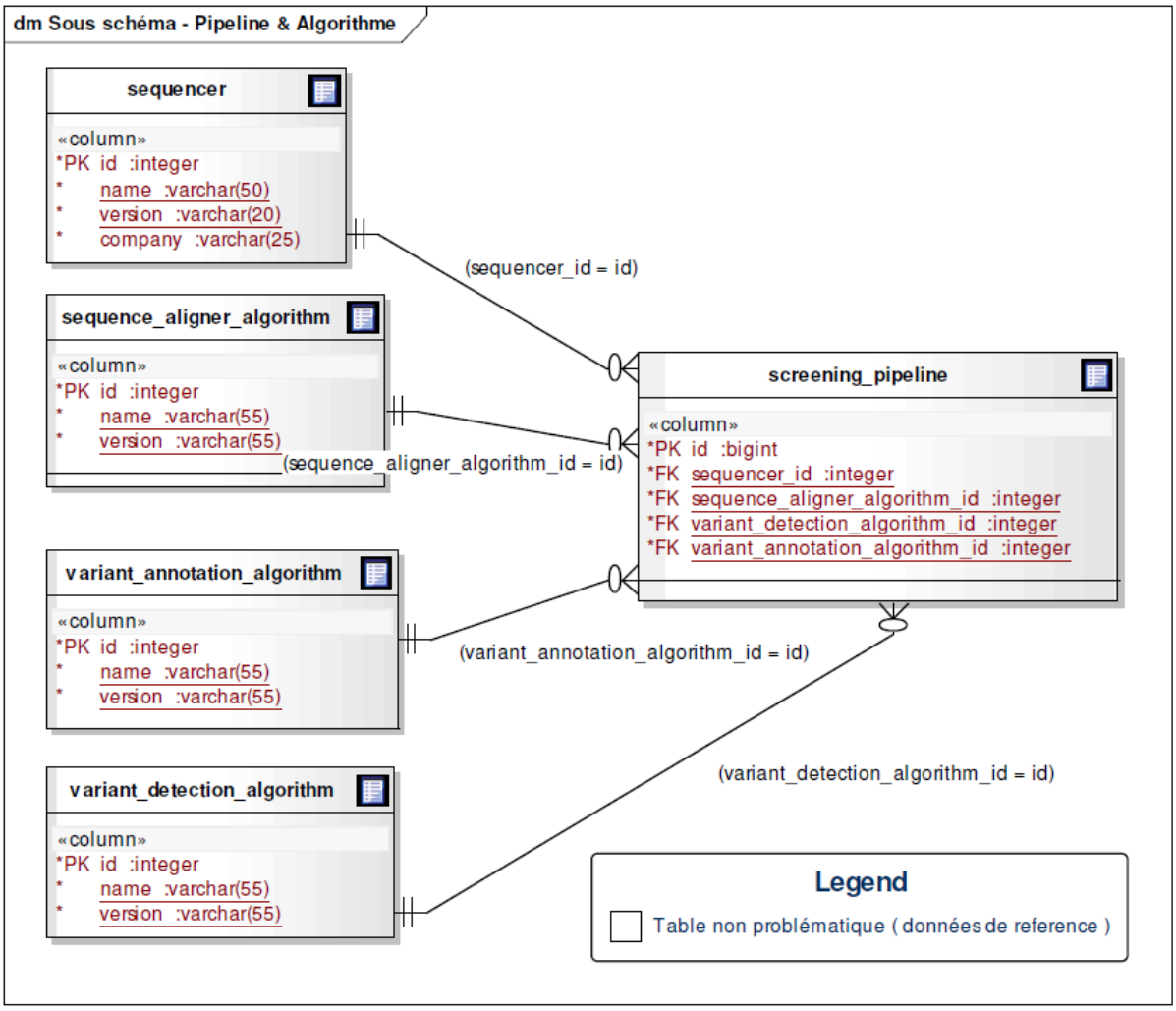


Figure 8 - Sous-schéma SQL #1

Le second sous-schéma est constitué des tables servant à décrire les gènes humains 18 et 19 ainsi que leurs relations avec les tables de chromosomes et de positions. Comme Mme Klos le mentionne dans son rapport, « Les tables “ngs_hg19_position” et “ngs_hg18_position” sont très volumineuses, par contre non référencées par les requêtes problématiques » (Klos, 2012). Par conséquent, elles ne sont pas candidates à une migration vers HBase dans le cadre de ce projet. Voici un diagramme illustrant le deuxième sous-schéma :

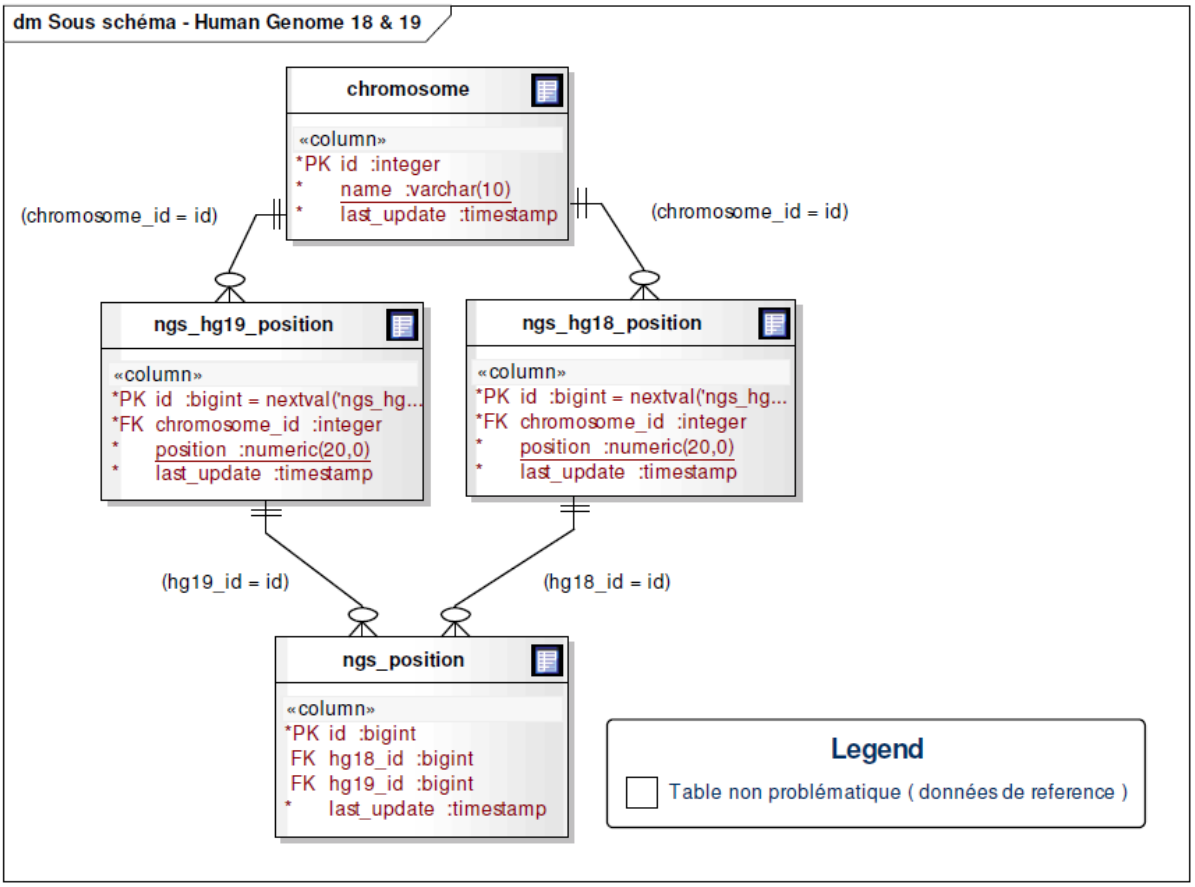


Figure 9 - Sous-schéma SQL #2

Le troisième et dernier sous-schéma contient plusieurs tables qui relient les données concernant les échantillons génomiques d'individus. Ce sous-schéma renferme notamment des informations concernant les variants des échantillons. Ces tables sont énormément volumineuses (plusieurs millions d'enregistrements) et font partie des tables recensées dans les requêtes problématiques. Ce sont donc ces tables qui seront transférées dans HBase. Ci-bas une illustration du dernier sous-schéma :

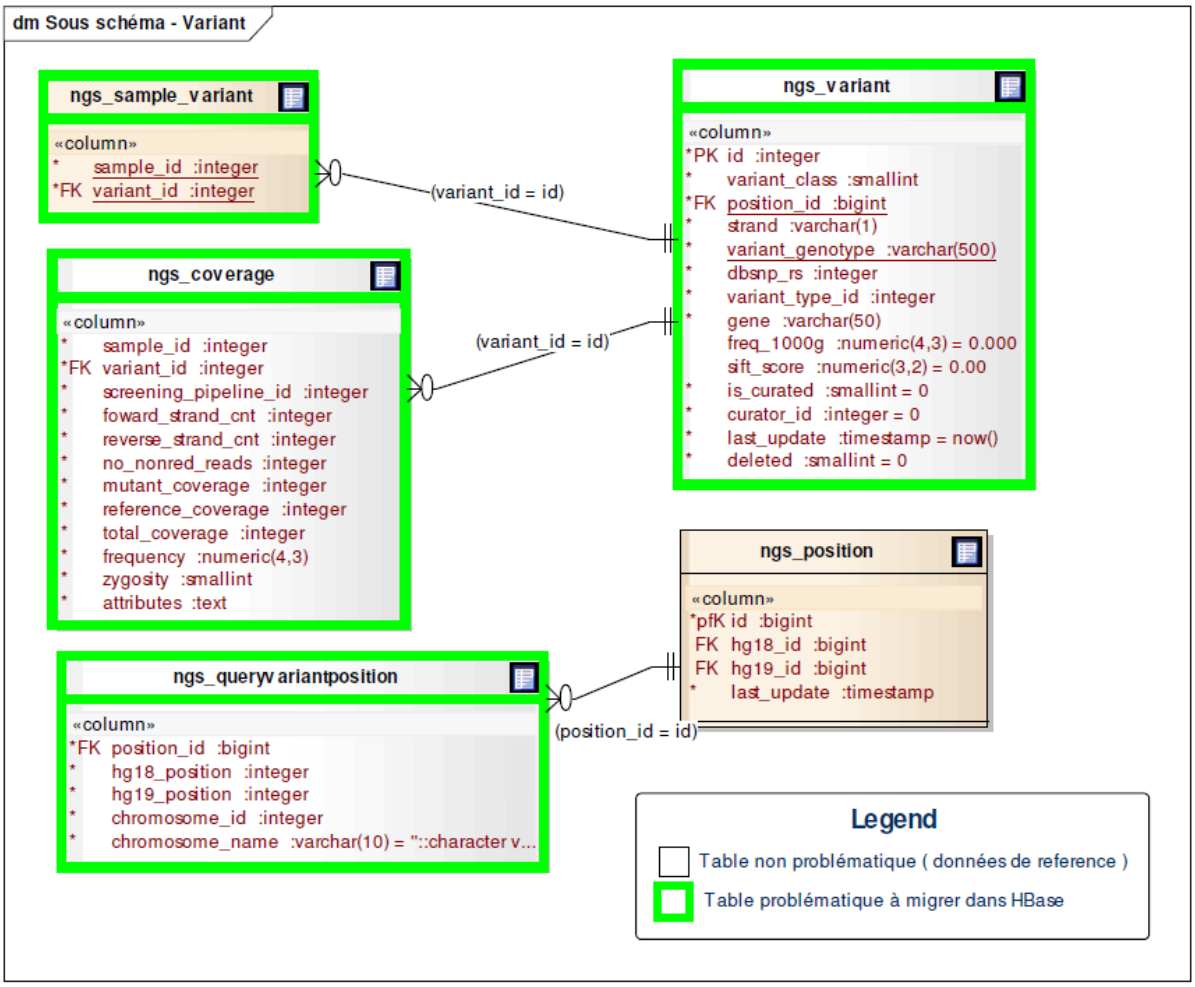
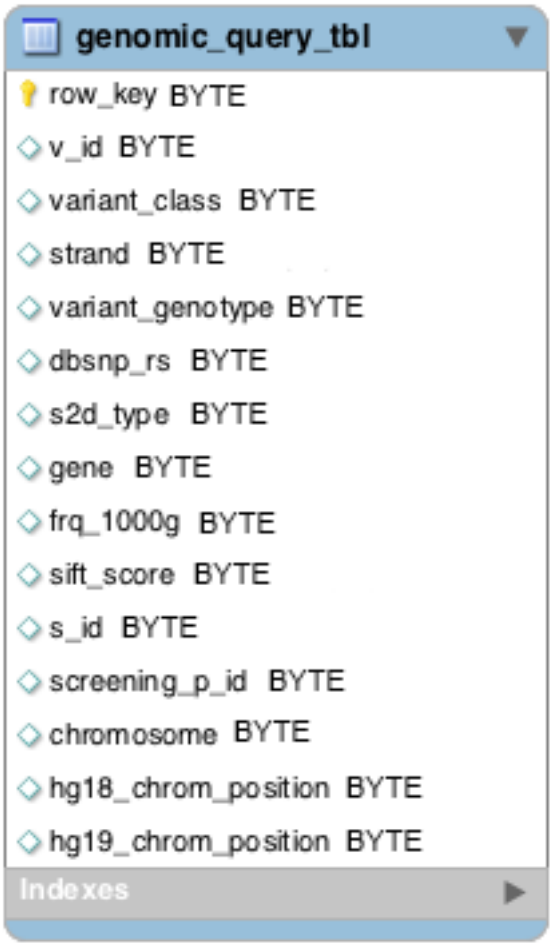


Figure 10 - Sous-schéma SQL problématique

4.2 Schéma HBase

Tout d'abord, il est important de noter que la base de données HBase dans ce projet est créée exclusivement pour apporter une solution à quelques requêtes problématiques dans le système actuellement utilisé par le CRCHUM. Le schéma est en fait constitué d'une grande table regroupant beaucoup d'informations, où chaque rangée est identifiée par une clé intelligente, constituée de plusieurs paramètres concaténés. La table regroupe certaines informations dupliquées là où nécessaire, comme le veut le modèle NoSQL. La structure de la table utilisée est définie par les extrants des requêtes problématiques. Or, chaque requête problématique sélectionne les mêmes informations, comme cela sera étudié à la prochaine

section, ce qui simplifie le tout. Autrement dit, les colonnes de la table représentent toutes les informations à extraire de la requête, alors que la clé représente une concaténation des divers paramètres pouvant varier lors de chaque requête. Voici la structure de la table dans la base de données HBase :



La clé pour chaque rangée possède le format suivant :

```
# Format de la clé de chaque rangée
variant_type_id | sample_id | pipeline_id | variant_id
```

Le caractère de séparation « | » entre les divers éléments de la clé permet de facilement effectuer des recherches dans celle-ci à l'aide d'expressions régulières. Voici un exemple concret de clé permettant de visualiser le format de chacune de ses sections :

```
# Exemple de clé de rangée
000011 | 000116 | 000022 | 00000000000001
```

Enfin, une seule famille de colonnes renfermera toutes les colonnes de la table, puisque chacune d'elle sera retournée pour chaque requête. Il n'y a ici aucun besoin de créer une segmentation au niveau des colonnes.

4.3 Transfert des données via Sqoop

Maintenant que Hadoop, HBase, PostgreSQL et Sqoop sont installés et fonctionnels, il est temps d'effectuer le transfert du modèle relationnel au modèle non relationnel. Bien entendu, rendu à cette étape, il faut que les données aient été chargées dans la base de données PostgreSQL. Cette étape triviale ne sera pas expliquée ici, puisque plusieurs moyens existent pour arriver à cette fin et ils sont tous aussi valables les uns que les autres. Les données génomiques du projet forment ensemble un amas de plusieurs gigaoctets distribués dans plusieurs fichiers SQL. Un fichier en particulier sert à créer le schéma et les tables, alors que tout le reste sert à effectuer les insertions de données.

Afin de sélectionner les bonnes données à transférer, Sqoop se servira d'une vue créée a pour cette opération. En effet, les colonnes à transporter de PostgreSQL vers HBase ne sont pas toutes contenues dans une même table. Elles sont divisées au travers de plusieurs tables partageant des relations 1..n et 1..1. Voici la vue utilisée dans le cadre de ce projet et qui est inspirée de celle produite par Mme Klos :

```
CREATE OR REPLACE VIEW "NGS_query_view" AS
      SELECT
            (
                  to_char(v.variant_type_id, '000000') ||
                  '|' ||
                  to_char(c.sample_id, '000000') ||
                  '|' ||
                  to_char(c.screening_pipeline_id, '000000') ||
                  '|' ||
                  to_char(v.id, '000000000000000')
            ) AS row_key,
            v.id AS v_id,
            v.variant_class,
            v.strand,
            v.variant_genotype,
            v.dbsnp_rs,
            v.variant_type_id AS s2d_type,
            v.gene,
            v.freq_1000g,
            v.sift_score,
            c.sample_id AS s_id,
            c.screening_pipeline_id AS screening_p_id,
            qvp.chromosome_name AS chromosome,
            qvp.hg18_position AS hg18_chrom_position,
            qvp.hg19_position AS hg19_chrom_position
      FROM
            ngs_variant v, ngs_coverage c, ngs_queryvariantposition
qvp
      WHERE
            v.position_id = qvp.position_id AND
            v.id = c.variant_id
      GROUP BY
            v.id, v.variant_genotype, v.dbsnp_rs, v.variant_type_id,
v.gene, v.freq_1000g, v.sift_score, qvp.chromosome_name,
qvp.hg18_position, qvp.hg19_position, c.sample_id,
c.screening_pipeline_id
;
```

Cette vue sélectionne des données à travers trois tables regroupe toutes les informations utiles aux requêtes problématiques. Il est aussi important de noter que c'est ici que le format de la clé de rangée utilisée dans HBase est créé.

Enfin, voici la commande à exécuter, en ligne de commande, afin de lancer la migration des données à l'aide de l'opération « import » de Sqoop :

```
$ sqoop import --connect jdbc:postgresql://localhost:5432/postgres
--username _postgres --query "select row_key, s2d_type, v_id,
variant_class, strand, variant_genotype, dbsnp_rs, gene,
freq_1000g, sift_score, chromosome, hg18_chrom_position,
hg19_chrom_position, s_id from \"NGS_query_view\" where
\$CONDITIONS" --hbase-table genomic_query_tbl --columns
row_key,s2d_type,v_id,variant_class,strand,variant_genotype,dbsnp_r
s,gene,freq_1000g,sift_score,chromosome,hg18_chrom_position,hg19_ch
rom_position,s_id --column-family cf_v_c_qvp --hbase-row-key
row_key --hbase-create-table --split-by row_key
```

Le transfert des millions de données prend une quinzaine de minutes à s'exécuter sur un disque dur SSD. Afin de bien comprendre les paramètres utilisés lors de la migration, chacun d'eux est expliqué dans la liste ci-dessous :

- --connect : indique à Sqoop de se connecter sur la base de données PostgreSQL à l'adresse *http://localhost:5432/postgres*.
- --username : spécifie l'utilisateur avec lequel se connecter à la base de données.
- --query : spécifie la requête à partir de laquelle la table HBase puisera ses données. Essentiellement, cette requête sélectionne toutes les colonnes de la vue créée précédemment.
- --hbase-table : spécifie une table HBase comme destination au lieu de HDFS.
- --columns : spécifie les colonnes à transférer vers HBase, y compris celle qui détiendra la clé de rangée.
- --column-family : spécifie dans quelle famille de colonne insérer les colonnes nouvellement ajoutées.
- --hbase-row-key : indique à HBase quelle colonne prendre comme clé de rangée.

- --hbase-create-table : indique à HBase de créer la table si elle est inexistante
- --split-by : spécifie la colonne à utiliser pour séparer les unités de travail. En d'autres mots, ce paramètre spécifie quelle colonne fera office de clé de rangée.

Si l'opération se termine avec un message de succès, la table HBase a été créée et est remplie de millions d'enregistrements. Une manière facile de vérifier cette affirmation est de se connecter à HBase et de vérifier l'existence de la table:

```
# NOTE: Hadoop et HBase doivent être démarrés!

$ hbase shell

hbase(main):001:0> list

TABLE
```

La table « genomic_query_tbl », créée par Sqoop, est bien présente. Il est possible de vérifier son contenu en effectuer un « scan » dessus, ce qui est l'équivalent d'effectuer un "SELECT * [...]" en SQL. Cependant, puisque la table contient plusieurs millions de rangées, la requête sera très, très longue à retourner étant donné qu'elle devra afficher les résultats à l'écran. Donc, pour vérifier les données, il suffit de faire un « scan » de la table et de l'interrompre avec la combinaison de touches CTRL+C. Voici une capture d'écran montrant un « scan » effectué et interrompu :

```
hbase(main):002:0> scan "genomic_query_tbl"
ROW                                                  COLUMN+CELL
 000001| 000039| 000022| 000000000003810              column=cf_v_c_qvp:chromosome, timestamp=1341185915849, value=1
 000001| 000039| 000022| 000000000003810              column=cf_v_c_qvp:dbsnp_rs, timestamp=1341185915849, value=9326115
 000001| 000039| 000022| 000000000003810              column=cf_v_c_qvp:freq_1000g, timestamp=1341185915849, value=1.000
 000001| 000039| 000022| 000000000003810              column=cf_v_c_qvp:gene, timestamp=1341185915849, value=C1orf173
 000001| 000039| 000022| 000000000003810              column=cf_v_c_qvp:hg18_chrom_position, timestamp=1341185915849, value=74809371
 000001| 000039| 000022| 000000000003810              column=cf_v_c_qvp:hg19_chrom_position, timestamp=1341185915849, value=0
 000001| 000039| 000022| 000000000003810              column=cf_v_c_qvp:s2d_type, timestamp=1341185915849, value=1
 000001| 000039| 000022| 000000000003810              column=cf_v_c_qvp:s_id, timestamp=1341185915849, value=39
 000001| 000039| 000022| 000000000003810              column=cf_v_c_qvp:sift_score, timestamp=1341185915849, value=0.00
 000001| 000039| 000022| 000000000003810              column=cf_v_c_qvp:strand, timestamp=1341185915849, value=
 000001| 000039| 000022| 000000000003810              column=cf_v_c_qvp:v_id, timestamp=1341185915849, value=3810
 000001| 000039| 000022| 000000000003810              column=cf_v_c_qvp:variant_class, timestamp=1341185915849, value=0
 000001| 000039| 000022| 000000000003810              column=cf_v_c_qvp:variant_genotype, timestamp=1341185915849, value=T/C
 000001| 000039| 000022| 000000000005017              column=cf_v_c_qvp:chromosome, timestamp=1341185947000, value=1
 000001| 000039| 000022| 000000000005017              column=cf_v_c_qvp:dbsnp_rs, timestamp=1341185947000, value=643457
 000001| 000039| 000022| 000000000005017              column=cf_v_c_qvp:freq_1000g, timestamp=1341185947000, value=1.000
 000001| 000039| 000022| 000000000005017              column=cf_v_c_qvp:gene, timestamp=1341185947000, value=PSRC1
 000001| 000039| 000022| 000000000005017              column=cf_v_c_qvp:hg18_chrom_position, timestamp=1341185947000, value=109626883
 000001| 000039| 000022| 000000000005017              column=cf_v_c_qvp:hg19_chrom_position, timestamp=1341185947000, value=0
 000001| 000039| 000022| 000000000005017              column=cf_v_c_qvp:s2d_type, timestamp=1341185947000, value=1
 000001| 000039| 000022| 000000000005017              column=cf_v_c_qvp:s_id, timestamp=1341185947000, value=39
 000001| 000039| 000022| 000000000005017              column=cf_v_c_qvp:sift_score, timestamp=1341185947000, value=0.00
 000001| 000039| 000022| 000000000005017              column=cf_v_c_qvp:strand, timestamp=1341185947000, value=
 000001| 000039| 000022| 000000000005017              column=cf_v_c_qvp:v_id, timestamp=1341185947000, value=5017
 000001| 000039| 000022| 000000000005017              column=cf_v_c_qvp:variant_class, timestamp=1341185947000, value=0
 000001| 000039| 000022| 000000000005017              column=cf_v_c_qvp:variant_genotype, timestamp=1341185947000, value=C/T
 000001| 000039| 000022| 000000000011899              column=cf_v_c_qvp:chromosome, timestamp=1341186027720, value=2
 000001| 000039| 000022| 000000000011899              column=cf_v_c_qvp:dbsnp_rs, timestamp=1341186027720, value=13394619
 000001| 000039| 000022| 000000000011899              column=cf_v_c_qvp:freq_1000g, timestamp=1341186027720, value=0.492
 000001| 000039| 000022| 000000000011899              column=cf_v_c_qvp:gene, timestamp=1341186027720, value=GREB1
 000001| 000039| 000022| 000000000011899              column=cf_v_c_qvp:hg18_chrom_position, timestamp=1341186027720, value=11644958
```

Chaque ligne représente une cellule, affichée à la droite de sa clé de rangée. Chaque cellule, à droite, est affichée dans un format « familleDeColonne : colonne », accompagnée de son estampille temporelle et de sa valeur.

CHAPITRE 5

REQUÊTE PROBLÉMATIQUE

5.1 Stratégie de recherche utilisée par le CRCHUM

Anna Klos, dans son rapport de fin d'études, mentionne que « les requêtes les plus problématiques sont celles qui cherchent à identifier les variants qui sont associés à un sous-ensemble d'échantillons et non-associés à un autre sous-ensemble d'échantillons selon les besoins de l'utilisateur qui interroge le système » (Klos, 2012).

Afin de mettre une image sur ces propos, voici le système S2D actuellement utilisé par le laboratoire Rouleau du CRCHUM :

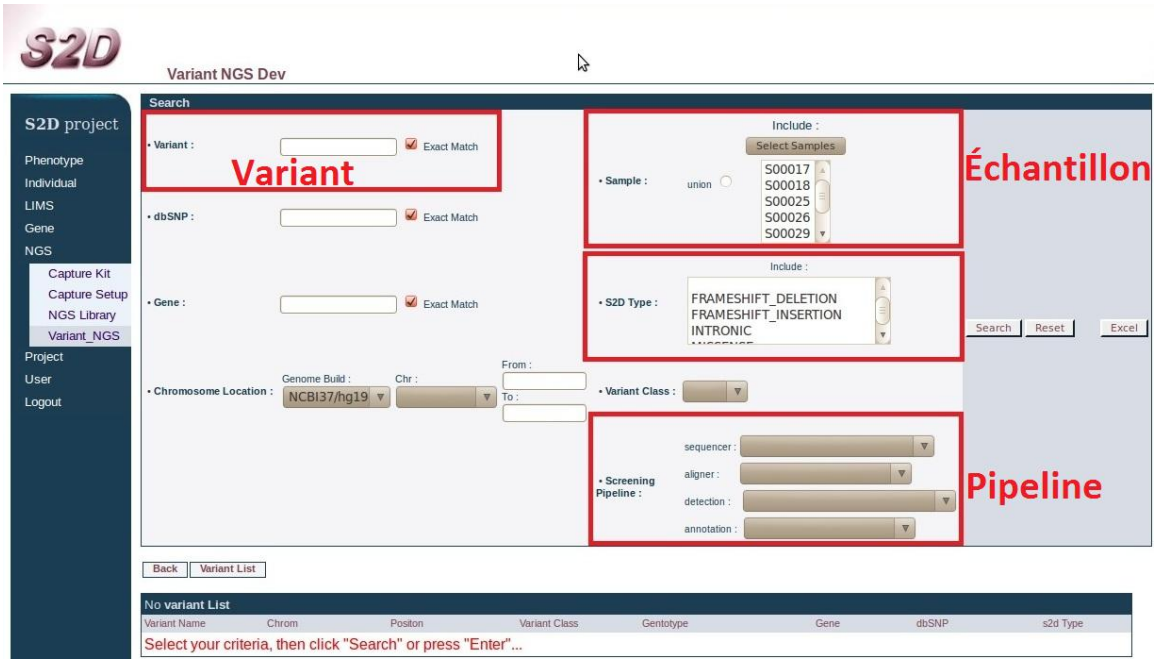


Figure 11 - Fenêtre de recherche du logiciel S2D

L'utilisateur a d'abord l'option d'exclure les variants présents dans des échantillons génomiques sélectionnés. En se servant des identifiants d'échantillons sélectionnés par l'utilisateur dans le formulaire, la requête suivante est utilisée en PostgreSQL :

Tableau 2 - Requête #1: Variant à exclure dans le résultat de recherche

```
SELECT DISTINCT variant_id
FROM NGS_sample_variant
WHERE sample_id in ( :SAMPLE_IDS_À_EXCLURE )
ORDER BY variant_id

L'application conserve dans une variable temporaire tous les variants
retournés par cette requête dans la variable "@excludeVariantsResults"
```

Le second choix qui s'offre à l'utilisateur consiste à inclure les variants présents dans des échantillons génomiques sélectionnés. Cependant, il est aussi possible d'émincer la recherche en précisant que les échantillons génomiques désirés doivent avoir été séquençés en utilisant un ou plusieurs « pipelines » de dépistage. Les deux variantes de cette requête SQL s'expriment comme suit :

Tableau 3 - Requête #2a: Variant à inclure dans le résultat de recherche en fonction du pipeline de dépistage

```
SELECT DISTINCT ngc.variant_id
FROM NGS_coverage ngc
WHERE ngc.sample_id in ( :SAMPLE_ID_À_INCLURE )
  AND screening_pipeline_id in ( :SELECTED_PIPELINE )
GROUP BY ngc.variant_id
ORDER BY ngc.variant_id

L'application rejette tous les variants qui sont présents dans la variable
"@excludeVariantsResults"
L'application conserve dans une variable temporaire tous les variants
retournés par cette requête dans la variable "@resultSet"
```

Tableau 4 - Requête #2b: Variant à inclure dans le résultat de recherche pour pour tout « pipeline » de dépistage

```
SELECT DISTINCT sv.variant_id
FROM NGS_sample_variant sv
WHERE sv.sample_id in ( :SAMPLE_ID_À_INCLURE )
GROUP BY sv.variant_id
ORDER BY sv.variant_id

L'application rejette tous les variants qui sont présents dans la variable
"@excludeVariantsResults"
L'application conserve dans une variable temporaire tous les variants
retournés par cette requête dans la variable "@resultSet"
```

Tous les résultats de ces requêtes s'inscrivent comme étapes intermédiaires à l'obtention des résultats finaux désirés. En effet, ces requêtes stockent leurs résultats dans une variable nommée « @resultSet », qui servira dans le cadre de la requête suivante afin de filtrer les résultats retournés à l'écran :

Tableau 5 - Requête #3: Requête utilisant la variable @resultSet

```
SELECT v.id, v.variant_class, v.strand, v.variant_genotype,
       v.dbSNP_rs, v.variant_type_id as s2d_type, v.gene,
       v.freq_1000g, v.Sift_score,
       qvp.chromosome_name as chromosome,
       qvp.hg18_position as hg18_chrom_position,
       qvp.hg19_position as hg19_chrom_position
FROM NGS_variant v, NGS_queryVariantPosition qvp
WHERE v.position_id = qvp.position_id
AND v.id in ( @resultSet )
AND v.variant_type_id = 4
GROUP BY v.id, v.variant_class, v.strand, v.variant_genotype,
       v.dbSNP_rs, v.variant_type_id, v.gene,
       v.freq_1000g, v.Sift_score,
       qvp.chromosome_name,
       qvp.hg18_position,
       qvp.hg19_position
```

Ainsi, l'un des objectifs de l'interface graphique illustrée précédemment est de fournir à l'utilisateur l'option d'inclure et d'exclure des échantillons afin d'inclure ou d'exclure tous les variants qui y sont associés. Ensuite, il est aussi possible d'exclure des variants en sélectionnant un ou plusieurs « pipelines » de dépistage. Avec ces variants en main, la requête finale est lancée et ses résultats sont retournés à l'utilisateur. Les résultats incluent plusieurs colonnes telles que l'identifiant du variant, sa classe, son génotype, son type, etc.

5.2 Stratégie de recherche à l'aide de HBase

Sachant que la requête du CRCHUM exprimée ci-haut sert essentiellement à inclure et exclure des variants, que ce soit directement ou via des échantillons, voici la stratégie de requête utilisée avec HBase comme base de données. Tout d'abord, qu'il soit d'ore et déjà mentionné que plus d'informations à ce propos se retrouveront dans le rapport de Jean-Philippe Bond, puisque sa manière de procéder fût retenue parmi toutes celles explorées.

Lors de l'inclusion de types de variants, d'échantillons, de « pipelines » de dépistage ou de variants précis, l'application créer un objet Scan et parcourra la table de la base de données créée pour l'occasion à la recherche d'enregistrements qui, dans leur clé de rangée, contiennent toutes les informations demandées. En guise de rappel, voici la composition de la clé de rangée de chaque enregistrement de la table HBase :

```
# Format de la clé de chaque rangée
variant_type_id | sample_id | pipeline_id | variant_id
```

L'application confirme qu'un enregistrement de données est conforme à la recherche effectuée en créant un objet Filter pour chaque condition (échantillon, « pipeline » et/ou variant) ajoutée à la requête. Ces objets Filter créeront un modèle d'expression régulière chacun qui sera comparé à chaque enregistrement de la table. Si un enregistrement parvient à traverser tous les filtres avec succès, alors ce dernier est considéré comme un résultat valide et est ajouté à la liste des résultats à retourner au client. Par exemple, considérons le scénario suivant :

> L'utilisateur désire avoir tous les variants dépistés par le « pipeline » de dépistage 24, qui font partie des échantillons 2001 à 3000 et qui ne font pas partie des échantillons 1001 à 2000.

Cet exemple créerait un filtre avec une expression régulière qui validerait le « pipeline » 24 dans le troisième tronçon de clé. De plus, mille filtres à expression régulière devraient être créés pour l'inclusion des échantillons 2001 à 3000 et mille autres pour l'exclusion des échantillons 1001 à 2000.

Cependant, cet exemple ne s'arrête pas simplement ici. Tel que spécifié lors de l'explication des requêtes effectuées sur le système actuel du laboratoire Rouleau, les échantillons exclus doivent faire en sorte que tous leurs variants soient aussi exclus de la recherche, même dans le cas où ces variants sont présents dans d'autres échantillons. Dans la solution développée, ceci se concrétise en une approche hybride. En effet, comme les relations échantillon-variant

sont documentées de façon concise dans la base de données PostgreSQL (table « ngs_sample_variant »), mais pas dans la table HBase, une requête est envoyée à la base de données PostgreSQL pour aller chercher tous les identifiants de variants à exclure en fonction des identifiants d'échantillons sélectionnés. Une fois que la réponse de PostgreSQL est disponible, tous les identifiants de variants reçus formeront chacun un nouveau filtre à expression régulière auquel seront confrontés les enregistrements. Ce mécanisme permet d'exclure des résultats tous les variants d'un ou plusieurs échantillons exclus dans le formulaire, sans pour autant transporter le schéma SQL en entier vers HBase.

Enfin, pour améliorer les performances, la table n'est pas parcourue séquentiellement, mais par intervalle. Effectivement, considérons l'exemple suivant :

> L'utilisateur sélectionne deux échantillons à inclure, par exemple 001056 et 406751. Ces échantillons produiront des clés comme suit (les autres sections de la clé sont purement inventées) :
>
> - `000004 | 001056 | 000024 | 000000000000050`
> - `000004 | 406751 | 000024 | 000000000000050`

S'il fallait parcourir la table séquentiellement, en gardant toujours en tête que HBase ordonne les rangées de manière alphanumérique en se basant sur la clé, il est facile de constater que ces deux clés sont à des milliers d'enregistrements l'une de l'autre. Pour éviter de balayer inutilement des dizaines de milliers de rangées, la recherche se fait par intervalle. Dès que les valeurs de deux échantillons (« samples ») sont supérieures à une limite, par exemple 500, l'application instanciera deux objets Scan qui effectueront chacun une recherche ultrarapide, au lieu d'effectuer une seule recherche qui serait beaucoup plus longue. Bref, l'exemple mentionné ci-haut produirait deux balayages très courts de la table, puisque les index de début et de fin de la recherche (« startRow » et « stopRow ») sont ajustés chacun à leur balayage respectif. Voir le rapport de Jean-Philippe Bond pour plus de détails à ce propos.

CHAPITRE 6

CLIENT WEB

Afin de démontrer la solution développée lors de la présentation orale, notre équipe a développé une application Web qui tente tant bien que mal d'imiter le formulaire de l'application S2D utilisée au laboratoire Guy Rouleau. L'application permet d'inclure des types de variant, des échantillons, des « pipelines » de séquençage, mais aussi d'exclure des échantillons. L'exclusion des échantillons a pour conséquence d'exclure tous les variants étant associés à ces échantillons de la recherche. L'application permet aussi de modifier en temps réel les paramètres de HBase tels que la grosseur du « batch », la grosseur de la cache ainsi que les valeurs des intervalles utilisés lors du balayage des données. Voici quelques captures d'écran de l'application Web développée :

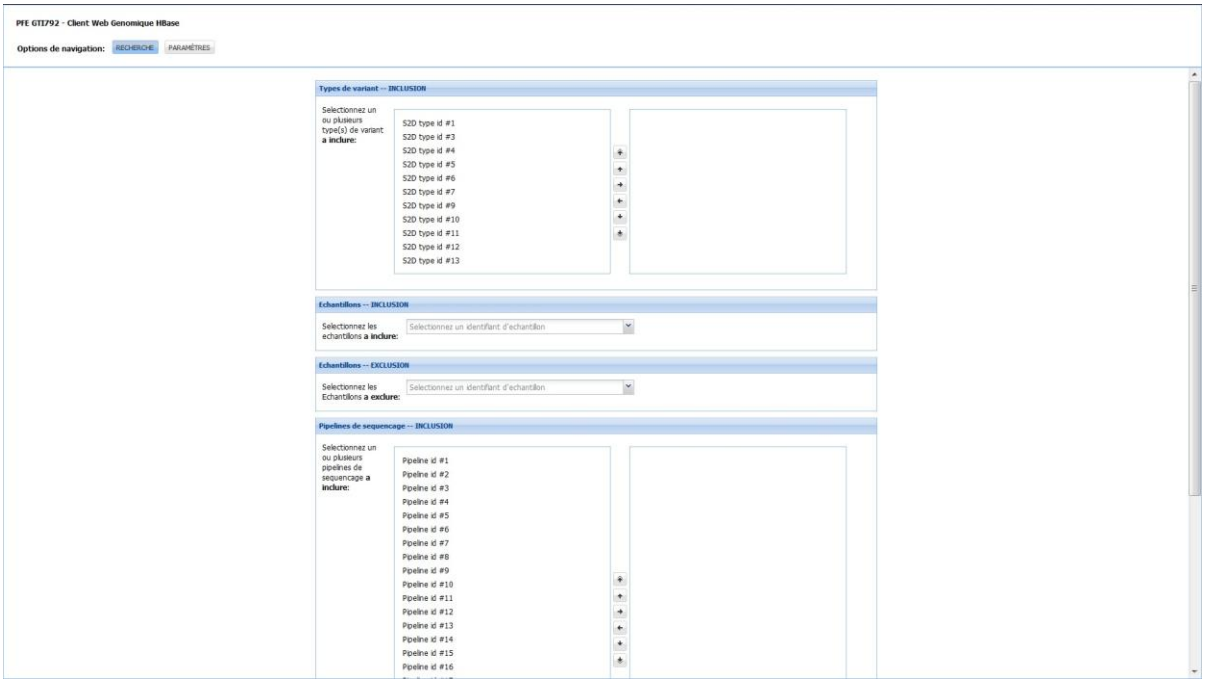


Figure 12 - Formulaire de recherche

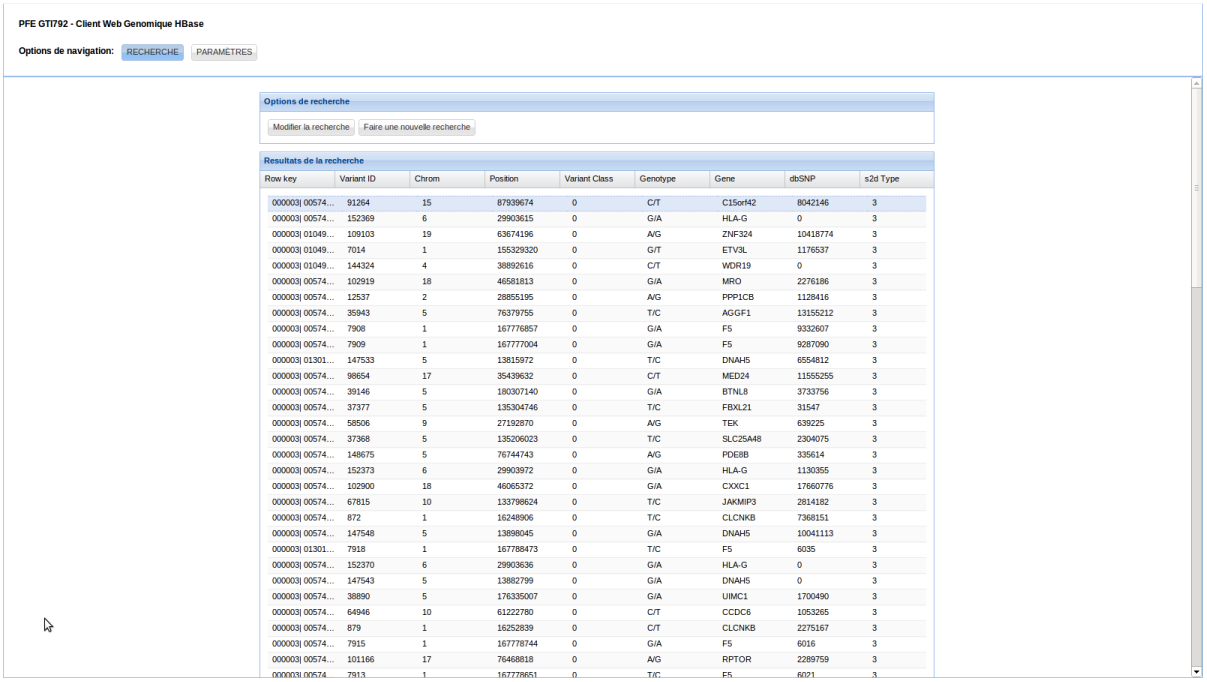


Figure 13: Page de résultats de la recherche

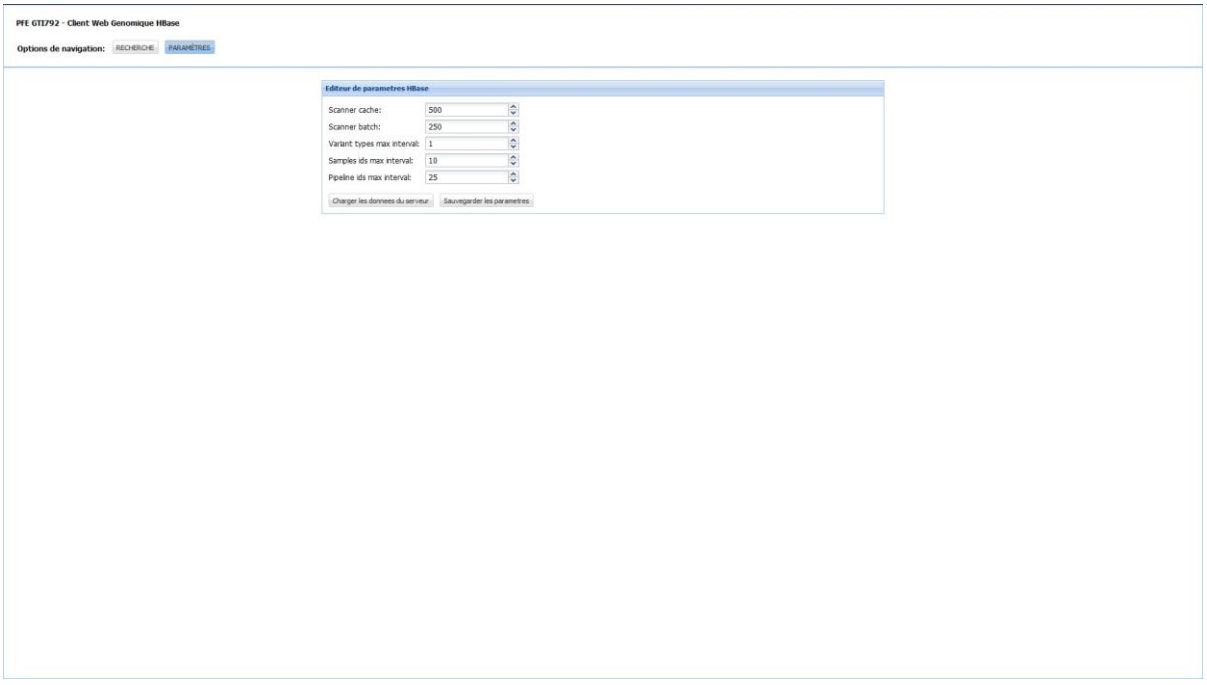


Figure 14: Formulaire de changements de paramètres HBase

L’application possède deux couches logicielles, soit la couche client et la couche serveur. Du côté client, ExtJS 4.1 est utilisé pour effectuer le rendu des éléments HTML (boutons, champs, tableaux, etc.) ainsi que pour la gestion des événements et appels au serveur. La couche applicative est assurée par une application Web propulsée par Spring MVC et utilise quelques nouveautés introduites dans le JDK 7. De plus, l’application utilise Hibernate pour gérer les interactions avec la base de données PostgreSQL et l'API de HBase pour se connecter au système de stockage pseudodistribué ou pleinement distribué. Enfin, le serveur applicatif sur lequel tourne la couche serveur est Apache Tomcat 7. Le cas d’utilisation typique se déroule comme suit :

> L’utilisateur personnalise sa recherche à l’aide des champs du formulaire. Il clique sur le bouton « Rechercher », qui envoie une requête POST au serveur dans le but de charger le magasin de données (« data store », terme utilisé dans le framework ExtJS) rattaché à la grille de résultats. Un observateur d’événement attend le chargement du magasin pour masquer le formulaire de recherche et afficher la grille remplie de résultats.

CONCLUSION

En conclusion, ce projet a réellement apporté son lot de défis et de difficultés. Partant d'un travail effectué par une ancienne étudiante de l'ÉTS, notre équipe a dû composer avec plusieurs embuches. Tout d'abord, les données qui nous ont été données en début de projet ne représentent qu'une partie des données. Il a été difficile pour nous de bien comprendre les liens entre les tables, puisque celles-ci n'étaient pas toutes présentes dans la base de données. Ensuite, la génomique étant un domaine scientifique qui utilise des termes techniques extrêmement compliqués, cet aspect a aussi contribué à freiner notre compréhension du problème initial. Par ailleurs, les technologies utilisées pour implémenter la solution étant émergentes et peu matures (Hadoop version 1.03 et HBase version 0.93), plusieurs problèmes ont fait surface, notamment lors de l'installation de la grappe et de HBase en mode pseudodistribué sur Mac OS X. En outre, trouver la meilleure stratégie de requête possible nous aura demandé beaucoup de tentatives avec diverses combinaisons d'approches et de paramètres. Somme toute, ce projet est, à nos yeux, une brillante réussite puisque tous les objectifs fixés dans le rapport d'étape et la proposition de projet ont été atteints et nous avons été en mesure de livrer un prototype fonctionnel qui permet l'exclusion d'échantillons génomiques. Bien entendu, les améliorations possibles sont infinies, tel que mentionné prochainement dans la section des recommandations, mais le travail accompli dans ce projet de fin d'études représente un bon pas en avant face à ce qui a été fait auparavant. Quoi qu'il en soit, ce prototype n'est nullement prêt à remplacer le système S2D du laboratoire Rouleau. Il représente par contre, selon nous, un très bon travail pratique à donner aux étudiants d'un futur cours portant sur le « big data ». Pour terminer, le « big data » est définitivement la voie vers laquelle les chercheurs en génétique s'orienteront dans un futur rapproché. Il n'en tient qu'à l'ÉTS à se tailler une place dans ce marché qui ne demande qu'à être conquis.

RECOMMANDATIONS

Tout d'abord, il faudrait utiliser la stratégie de Scan développée par notre équipe dans un contexte MapReduce optimisé. Plus précisément, il faudrait utiliser une « job » Map par objet Scan (avec filtre associé) créé par l'application et lancer le tout en parallèle dans un environnement distribué comportant plus de trois nœuds. Le Reducer serait utilisé pour effectuer les exclusions de variants en fonction des échantillons.

Ensuite, il faudrait revoir le schéma de données du laboratoire Guy Rouleau dans son ensemble. Nous avons uniquement eu un ensemble de données limité datant de l'an dernier sur lequel travailler. Selon ce que M. Ousmane Diallo nous a mentionné lors de notre visite au laboratoire, le schéma SQL du CRCHUM a changé depuis. Il faudrait donc réviser entièrement le schéma HBase sur réception du schéma SQL complet du laboratoire Rouleau. L'objectif de cette opération serait de valider que le schéma HBase développé convienne réellement aux besoins actuels du laboratoire et qu'il est intègre par rapport au schéma actuellement utilisé. Il faudrait, si nécessaire, modifier le schéma HBase pour qu'il reflète les derniers changements du CRCHUM.

Il serait aussi souhaitable de revoir la composition de la clé de rangée selon l'occurrence des paramètres de recherche. En effet, la composition de la clé est très importante, puisqu'elle définit l'ordonnancement automatique et continuellement effectué par HBase. Avec le peu d'information que nous disposons sur les requêtes effectuées par les employés du laboratoire Guy Rouleau, nous sommes loin d'être certains que la composition actuelle de la clé concatène de manière optimale les paramètres de recherche.

Par ailleurs, il serait intéressant de montrer le travail accompli au laboratoire du CRCHUM afin de récolter une rétroaction de leur part. De cette façon, il serait possible de connaitre leur opinion sur les fonctionnalités de l'application, les requêtes effectuées ainsi que les résultats présentés. Avec la rétroaction du client, il serait possible de produire de nouveaux requis,

qu'ils soient logiciels ou non, et d'itérer dessus afin d'améliorer continuellement le prototype et peut-être le voir un jour remplacer le système S2D actuel.

Afin de comparer les produits disponibles sur le marché, il serait pertinent d'installer et de tester les performances d'autres systèmes de base de données distribuées. Par exemple, il serait possible de reproduire le même genre de solution, mais avec HyperTable ou Cassandra. Ensuite, il faudrait analyser les performances de ces systèmes, les comparer à HBase et les classer selon la complexité de mise en place. Puisque ce projet a aussi pour but de fournir un travail pratique à un futur cours de l'ÉTS, la solution ne doit pas être excessivement complexe à mettre en place.

Enfin, il n'y a de limites que celles que l'on s'impose lorsqu'il est question d'amélioration logicielle. Les possibilités d'amélioration d'un tel système sont infinies.

LISTE DE RÉFÉRENCES

Apache. (2012, mai 8). *MapReduce Tutorial*. Consulté le juin 2, 2012, sur hadoop: http://hadoop.apache.org/common/docs/r1.0.3/mapred_tutorial.html

Apache Software Foundation. (2012, mai 6). *Apache Hadoop 2*. Consulté le juillet 4, 2012, sur Hadoop: http://hadoop.apache.org/common/docs/current/

Apache Software Foundation. (2012, 08 août). *Apache HBase Reference Guide* . Consulté le juin 22, 2012, sur Apache HBase: http://hbase.apache.org/book.html#columnfamily

Apache Software Foundation. (2012, août 1). *Apache Sqoop*. Consulté le juillet 11, 2012, sur Apache Sqoop: http://sqoop.apache.org/

Apache Software Foundation. (2012, mai 8). *Cluster Setup*. Consulté le juin 20, 2012, sur Hadoop: http://hadoop.apache.org/common/docs/r1.0.3/cluster_setup.html

Borthakur, D. (2009, septembre 1). *HDFS Architecture*. Consulté le juin 14, 2012, sur hadoop: http://hadoop.apache.org/common/docs/r0.20.1/hdfs_design.html

George, L. (2011). *HBase: The Definitive Guide.* Sebastopol: O'Reilly Media.

Klos, A. (2012). *Optimisation de recherche grâce à HBase sous Hadoop.* Montréal: École de technologie supérieure.

Laboratoire Guy Rouleau. (2012). *Projet S2D*. Consulté le juin 6, 2012, sur Laboratoire Guy Rouleau: http://www.laboguyrouleau.ca/S2D.html

McKenna, A. (2012, mai 22). *Exit le nuage informatique, l'heure est au «big data»*. Consulté le juin 4, 2012, sur La Presse: http://techno.lapresse.ca/nouvelles/201205/22/01-4527312-exit-le-nuage-informatique-lheure-est-au-big-data.php

PostgreSQL. (2010, octobre 20). *17.1. The PostgreSQL User Account*. Consulté le juillet 9, 2012, sur PostgreSQL 9.0.8 Documentation: http://www.postgresql.org/docs/9.0/interactive/postgres-user.html

Saleh, T. (s.d.). *Installing PostgreSQL for Rails 3.1 on Lion*. Consulté le juillet 8, 2012, sur Tammer Saleh: http://tammersaleh.com/posts/installing-postgresql-for-rails-3-1-on-lion

Wikipédia. (2012, juillet 20). *Big data*. Consulté le juin 4, 2012, sur Wikipédia: http://fr.wikipedia.org/wiki/Big_data

Wikipédia. (2012, août 1). *Hadoop*. Consulté le juin 29, 2012, sur Wikipédia: http://fr.wikipedia.org/wiki/Hadoop#ZooKeeper

Wikipédia. (2012, mars 29). *MapReduce*. Consulté le juin 2, 2012, sur Wikipédia: http://fr.wikipedia.org/wiki/MapReduce

Wikipédia. (2012, juillet 25). *NoSQL*. Consulté le juin 15, 2012, sur Wikipédia: http://fr.wikipedia.org/wiki/Nosql

BIBLIOGRAPHIE

Apache. (2012, mai 8). *MapReduce Tutorial*. Consulté le juin 2, 2012, sur hadoop: http://hadoop.apache.org/common/docs/r1.0.3/mapred_tutorial.html

Apache Software Foundation. (2012, mai 6). *Apache Hadoop 2*. Consulté le juillet 4, 2012, sur Hadoop: http://hadoop.apache.org/common/docs/current/

Apache Software Foundation. (2012, 08 août). *Apache HBase Reference Guide* . Consulté le juin 22, 2012, sur Apache HBase: http://hbase.apache.org/book.html#columnfamily

Apache Software Foundation. (2012, août 1). *Apache Sqoop*. Consulté le juillet 11, 2012, sur Apache Sqoop: http://sqoop.apache.org/

Apache Software Foundation. (2012, mai 8). *Cluster Setup*. Consulté le juin 20, 2012, sur Hadoop: http://hadoop.apache.org/common/docs/r1.0.3/cluster_setup.html

Apache Software Foundation. (2012, août 8). *HBase ACID Semantics*. Consulté le juillet 3, 2012, sur Apache HBase: http://hbase.apache.org/acid-semantics.html

Borthakur, D. (2009, septembre 1). *HDFS Architecture*. Consulté le juin 14, 2012, sur hadoop: http://hadoop.apache.org/common/docs/r0.20.1/hdfs_design.html

Brandeis University. (2008). *Introduction to Hadoop*. Consulté le juillet 10, 2012, sur Computer Science Department: http://pages.cs.brandeis.edu/~cs147a/lab/hadoop-intro/

Cloudera. (s.d.). *Hadoop Overview*. Consulté le juin 9, 2012, sur Cloudera: http://www.cloudera.com/what-is-hadoop/hadoop-overview/

George, L. (2011). *HBase: The Definitive Guide.* Sebastopol: O'Reilly Media.

Jeanson, A. (2011, mars 10). *Comprendre Hadoop en moins de 5 minutes*. Consulté le juin 19, 2012, sur Java EE Performance: http://www.opensides.fr/2011/03/10/hadoop-en-moins-de-5-minutes/

Klos, A. (2012). *Optimisation de recherche grâce à HBase sous Hadoop.* Montréal: École de technologie supérieure.

Laboratoire Guy Rouleau. (2012). *Projet S2D*. Consulté le juin 6, 2012, sur Laboratoire Guy Rouleau: http://www.laboguyrouleau.ca/S2D.html

McKenna, A. (2012, mai 22). *Exit le nuage informatique, l'heure est au «big data»*. Consulté le juin 4, 2012, sur La Presse: http://techno.lapresse.ca/nouvelles/201205/22/01-4527312-exit-le-nuage-informatique-lheure-est-au-big-data.php

PostgreSQL. (2010, octobre 20). *17.1. The PostgreSQL User Account*. Consulté le juillet 9, 2012, sur PostgreSQL 9.0.8 Documentation: http://www.postgresql.org/docs/9.0/interactive/postgres-user.html

Saleh, T. (s.d.). *Installing PostgreSQL for Rails 3.1 on Lion*. Consulté le juillet 8, 2012, sur Tammer Saleh: http://tammersaleh.com/posts/installing-postgresql-for-rails-3-1-on-lion

Wikipédia. (2012, juillet 20). *Big data*. Consulté le juin 4, 2012, sur Wikipédia: http://fr.wikipedia.org/wiki/Big_data

Wikipédia. (2012, août 1). *Hadoop*. Consulté le juin 29, 2012, sur Wikipédia: http://fr.wikipedia.org/wiki/Hadoop#ZooKeeper

Wikipédia. (2012, mars 29). *MapReduce*. Consulté le juin 2, 2012, sur Wikipédia: http://fr.wikipedia.org/wiki/MapReduce

Wikipédia. (2012, juillet 25). *NoSQL*. Consulté le juin 15, 2012, sur Wikipédia: http://fr.wikipedia.org/wiki/Nosql