Université du Québec
École de technologie supérieure

RAPPORT TECHNIQUE
PRÉSENTÉ À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
DANS LE CADRE DU COURS LOG792 PROJET DE FIN D'ÉTUDES EN GÉNIE LOGICIEL

OPTIMISATION DE RECHERCHE GRÂCE À HBASE SOUS HADOOP

ANNA KLOS
KLOA22597907

DÉPARTEMENT DE GÉNIE LOGICIEL ET DES TI

Professeur superviseur

ALAIN APRIL

MONTRÉAL, 10 AVRIL 2012
HIVER 2012

REMERCIEMENTS

Alain April : Professeur de Génie Logiciel et membre du laboratoire de recherche en génie logiciel à l'ÉTS.

Abraham II Gómez Morales : Étudiant au doctorat en « *Cloud Computing* » à l'ÉTS.

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

Luis Eduardo Bautista Villalpando : Étudiant au doctorat à l'ÉTS.

OPTIMISATION DE RECHERCHE GRÂCE À HBASE SOUS HADOOP

ANNA KLOS
KLOA22597907

RÉSUMÉ

Les environnements de développement pour le projet de l'informatique de nuage sont difficiles à monter faute de manque de documentation, stabilité de leurs composantes logicielles et de leurs complexités.

Dans le but de diminuer l'impact de ces contraintes et de faciliter l'accès à un tel environnement, il a été décidé d'utiliser un échantillon des données d'un de centres de recherches en génomiques et de monter un tel environnement.

Le problème que ce projet tente de résoudre est de créer un environnement facile à utiliser par des étudiants dans le but de se familiariser avec l'informatique de nuage et des base de donnée non relationnelle tel que HBase.

Les résultats obtenus peuvent être utilisés pour un projet futur dans lequel une application client peut être développée pour traduire des requetés SQL vers des requêtes No-SQL et valider si un model non-relationnel est adéquat ou pas.

Mots-clés : Base de données relationnelles orienté objet (PostgrSQL), Base de données géantes non-relationnelle (HBase), Cloudera, Hadoop, HDFS, MapReduce, Sqoop.

Table des matières

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

CDH	Distribution de Hadoop par Cloudera « Cloudera's Distribution Including Apache Hadoop »
CRCHUM	Centre Hospitalier de l'Université de Montréal
CRUD	Les opérations de manipulation de données « Create, Read, Update, Delete »
HBase	Base de données de Hadoop « Hadoop Data Base »
HDFS	Système distribué de gestion de fichiers « Hadoop Distributed File System »
JVM	Machine virtuelle de Java « Java Virtual Machine »
RDBMS	Système de gestion de base de données relationnelle « Relational data base management system »
SSH	Protocole de communication sécurisé « Secure Shell »
Sqoop	SQL vers Hadoop « SQL-to-Hadoop »
VPN	Réseau privé virtuel « Virtual private network »

LISTE DES SYMBOLES ET UNITÉS DE MESURE

MB	Mégaoctet « Mégabyte »
TB	Téraoctet « Terabyte »

INTRODUCTION

Les environnements de développement pour le projet de l'informatique de nuage sont difficiles à monter faute de manque de documentation, de stabilité de leurs composantes logicielles et de leurs complexités.

Dans le but de diminuer l'impact de ces contraintes et de faciliter l'accès à un tel environnement, il a été décidé de monter un environnement de développement avec Hadoop et HBase. De plus pour démontrer les différentes caractéristiques d'une base de données non relationnelle qui est HBase, un échantillon des données de la génomique des patients du Centre Hospitalier de l'Université de Montréal (CRCHUM) est utilisé. Cet échantillon réel permet de démontrer la puissance de cette base de données ainsi que la transformation d'une structure de données relationnelles vers une structure non relationnelle en répondant aux mêmes exigences fonctionnelles.

Le problème que ce projet tente de résoudre est de créer un environnement facile à utiliser par des étudiants dans le but de se familiariser avec l'informatique de nuage et des base de donnée non relationnelle tel que HBase.

Ce projet a comme objectif principal la création d'un environnement « type » sous Hadoop avec Cloudera dans le but de le réutiliser pour le nouveau cours de M. Alain April au sujet de « Cloud Computing » et « Hadoop ». Un sous-ensemble de données (problématiques) de CRCHUM pour des études génomiques sera utilisé dans le but de monter une base de données HBase sous Hadoop. L'identification des données problématiques se fera en analysant la stratégie utilisé par CRCHUM lors de l'interrogation de la liste de variant de la base de donnée de séquençage à haut débit (NGS).

Suite à cette analyse les tables utilisé dans l'interrogation de la liste de variant seront transformé sous le format de HBase. La traduction des requêtes SQL problématiques sera faite pour démontrer le fonctionnement d'une base de données non relationnel.

CHAPITRE 1

REVUE DE LA LITTÉRATURE

1.1 Projet Hadoop d'Apache

Le projet Hadoop d'Apache est un projet open source qui regroupent une variété de sous-projet qui permettent de bâtir des solutions applicatives pour l'informatique distribué. Les membres de la communauté de Hadoop sont tous des bénévoles. C'est pour cette raison qu'il n'est pas possible d'obtenir du support immédiat pour leurs solutions. Par contre le code source de n'importe laquelle de leurs solutions est complètement ouvert et disponibles pour toutes corrections.

1.1.1 HDFS : Système distribué de gestion de fichiers

« Hadoop Distributed File System » (HDFS) est le principal système de gestion distribué de très gros fichiers, utilisé par les applications développées dans le cadre du projet Hadoop. HDFS est le système le plus testé et utilisé en production. Il est très populaire car il incorpore la réplication des données, il est très tolérant aux erreurs et est extensible.

1.1.2 Hadoop MapReduce : Framework de transformation de données

Hadoop MapReduce est un « Framework » Java, qui permet de traiter des énormes quantités des données en parallèle. Son architecture est entièrement basé sur celle de MapReduce de Google, laquelle a était publié en 2003 dans le document intitulé « MapReduce : Simplified Data Processing on Large Clusters ».
MapReduce est composé des « MapReduce *Job* », les quelles sont composé des deux types de tâches suivantes et qui s'exécutent en parallèle:

- « Map Task »;
- « Reduce Task ».

Le « Framework » de MapReduce s'occupe de la gestion, de surveillance et de réexécution de toutes tâches qui n'a pas été complétée avec succès.

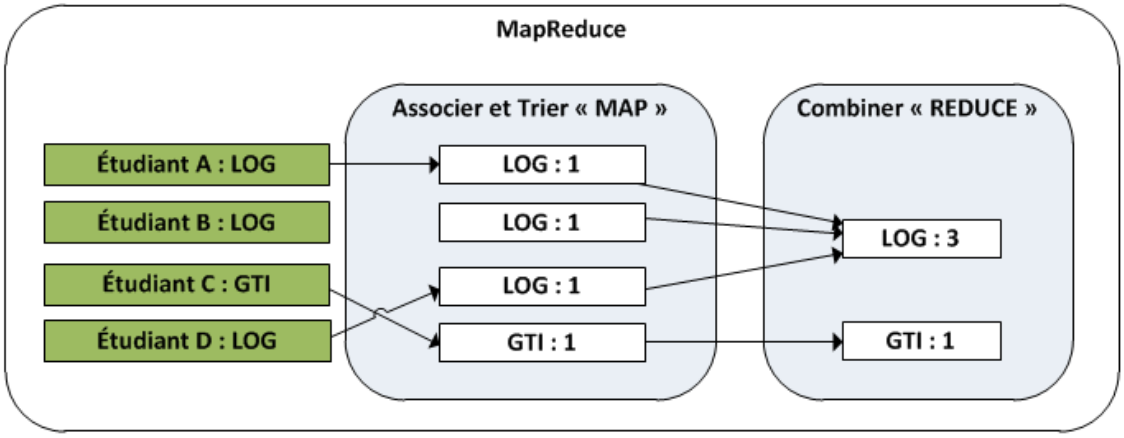


Figure 1 : Model de programmation "MapReduce"

1.2 HBase : Base de données de Hadoop

HBase est une base de données distribué, non-relationnelle et open-source d'Apache. Elle a était créée au début de 2007, par la compagnie « Powerset » situé à San Francisco, en s'inspirant du papier de Google sur BigTable. C'est le système distribué de gestion de fichiers de Hadoop (HFDS) qui à la base de HBase dans le but d'entreposer et d'indexer ses données, tout comme GFS est à la base de BigTable.

1.2.1 La couche de persistance

La couche de persistance de HBase est composée de trois éléments principaux suivants :

- Un serveur maître « Master server »;
- Un ou plusieurs serveurs de région « Region servers » ;
- Une librairie client « Client API ».

Le serveur maître est responsable d'assigner les régions au serveur de régions dans le but d'équilibrer leurs charges. Cette tâche d'assignation des régions est faite grâce au système distribué de coordination ZooKeeper. Ce système de coordination est la composante clé de

HBase. C'est aussi grâce à elle que HBase peut s'assurer qu'un seul serveur maître est en opération à un moment donnée. Le serveur maître maintient aussi l'état du cluster et assure l'équilibre de charges entre les serveurs de régions. De plus il s'occupe de tous les changements au niveau du modèle de données. C'est pour cette raison qu'il est très peu occupé. Il est à noter que le serveur maître ne communique jamais avec le client.

Les serveurs de régions sont quant à eux responsables de la transmission de toutes les requêtes d'écritures et de lectures vers les régions dont ils sont propriétaire. Ces requêtes sont communiquées directement aux serveurs de régions par le(s) client(s).

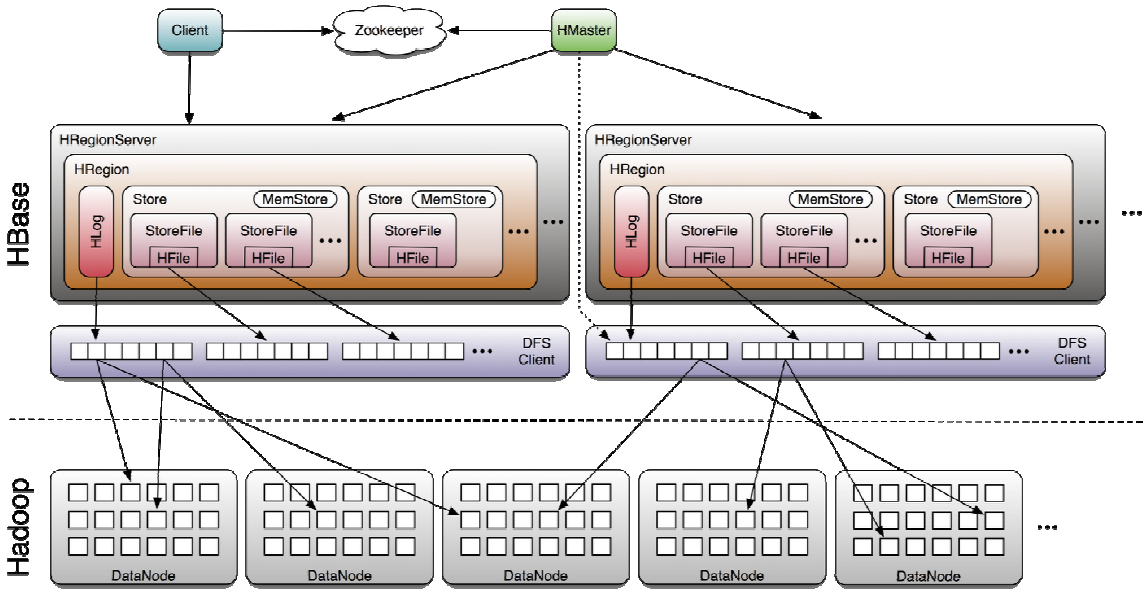


Figure 2: La plateforme HBase

Tirée de HBase:Architecture » New IT Farmer (7, déc. 2011)

1.2.2 Les modes d'exécution

Il est possible d'exécuter HBase dans un de deux modes suivants :

- Autonome « Standalone »;
- Distribué « Distributed ».

Le mode autonome est le mode par défaut de HBase, c'est le mode simpliste et il s'adresse à toute utilisation à des fins strictement personnelles et éducatives. Ce mode n'utilise pas le

système distribué de gestion de fichiers HDFS et exécute toutes les composantes nécessaire à HBase dans un seul processus de la machine virtuelle de Java (JVM).
Le mode distribué utilise le système de gestion de fichiers HDFS et se subdivise en deux sous-modes suivants :

- Pseudo-distribué;
- Entièrement distribué.

Le premier sous-mode appelé « pseudo-distribué » utilise HDFS et s'exécute que sur un seul serveur. Par conséquent il est conseillé de l'utiliser qu'à des fins de prototypage et de test. Il ne fait surtout pas l'utiliser en production ou bien pour évaluer la performance de HBase.
Le deuxième sous-mode appelé « entièrement distribué » utilise toute la puissance de HDFS et est utilisé en production. Par contre il est beaucoup plus difficile à configurer est exigé un minimum des deux serveurs.

1.2.3 Le modèle de données

La base de données HBase est composée de tables, de familles de colonnes, de colonnes, d'enregistrement, de cellules et de version de colonnes. C'est un modèle non-relationnel et basé entièrement sur l'architecture d'entreposage de BigTable. Dans plusieurs ouvrage ce type de base données est appelé « Columnar DataBase » ou modèle vertical.
Pour définir un schéma dans HBase il faut à la base définir les tables ainsi que les familles de colonnes. Les familles de colonnes permettent de regrouper les données de la même façon que les tables. Tout ceci dans le but de faciliter l'accès aux données via clé primaire car il n'est pas possibles de faire de requêtes sur plusieurs tables en même temps comme dans une base de données relationnelle. Les colonnes sont composées de cellules et de leurs valeurs. Une notion intéressante dans HBase est qu'une dimension additionnelle est sauvegardée et c'est celle de versions de colonnes. À part les tables et les familles de colonnes, les colonnes sont définies lors d'insertion de données. C'est pour ça que c'est une structure complètement extensible en mode d'exécution. Une table de HBase est enregistrée dans un ou plusieurs fichiers qui peuvent être distribués sur un ou plusieurs serveurs.

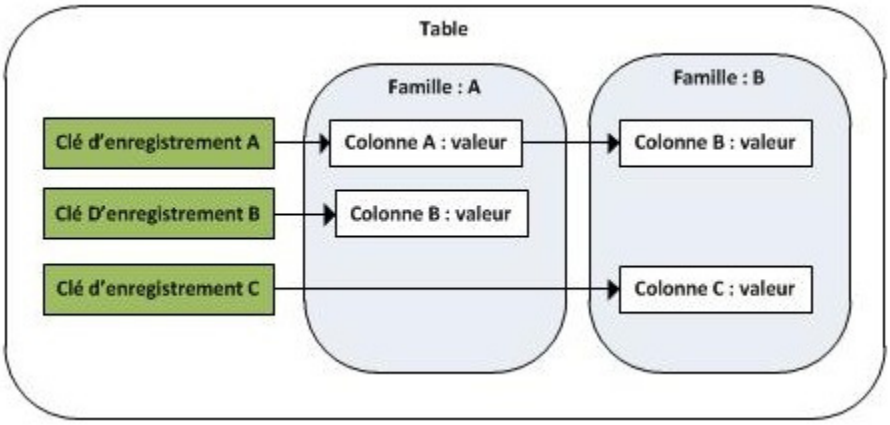


Figure 3 : Model de données

Inspiré de (George, 2011)

Il est très important de bien concevoir la taxonomie de clés dans HBase. Puisque dépendamment comment ces derniers sont conçus, il sera plus facile de contrôler comment et où les données sont sauvegardées. Aussi il sera plus facile d'accéder aux données. Il est à noter que les enregistrements dans HBase sont toujours ordonnés selon la valeur lexicographique, c.-à-d. en ordre alphabétique.

Tableau 1 : L'ordre d'enregistrement dans un fichier de HBase

Clé « Row Key »
A-1
A-10
A-2
A-a

1.2.4 Les opérations supporté par HBase

Toutes les opérations de base d'une base de données relationnelle sont aussi supportées par HBase :

- Ajout et modification de données « PUT » ;
- Lecture de données « GET » ;
- Suppression de données « DELETE » ;

- Recherche de données « SCAN ».

1.2.4.1 L'opération d'ajout et de modification « PUT »

L'instruction qui permet de sauvegarder les données dans HBase est celle appelée « PUT ». Elle permet de sauvegarder une nouvelle version d'une colonne avec une nouvelle valeur à l'intérieur d'une table à l'adresse d'une clé, d'une famille de colonne et d'une colonne.

Tableau 2 : Instructions de sauvegarde de données

Inspiré de (HBase/Shell - Hadoop Wiki)

```
Pour enregistré une première version d'une colonne avec la valeur "1234"
dans la table « T_A », dans la colonne « C_A » de la famille de colonnes
« CF_A » à la clé « ROW_A », il faut exécuter la commande suivante :
          hbase> put 'T_A', 'ROW_A', 'CF_A:C_A', '1234'
Si la colonne "C_A" n'existe pas dans la famille "CF_A", elle sera
automatiquement crée.

Pour enregistré une deuxième version d'une colonne avec la valeur "5678"
dans la même table et la même colonne qu'au exemple précèdent, il faut
exécuter la commande suivante :
          hbase> put 'T_A', 'ROW_A', 'CF_A:C_A', '5678'
Une nouvelle version de la colonne "CF_A :C_A" est enregistré. Comme il
n'est pas nécessaire de vérifier si l'enregistrement existe déjà dans la
table, les opérations d'écritures sont très performantes.
```

1.2.4.2 L'opération de lecture « GET »

L'instruction qui permet de lire les données précédemment sauvegardé dans HBase est celle appelée « GET ». Elle permet de lire le contenu d'un enregistrement ou bien d'une ou plusieurs familles de colonne à l'adresse d'une clé donnée.

Tableau 3 : Instruction de lecture de données

Inspiré de (HBase/Shell - Hadoop Wiki)

```
Pour lire le contenu d'un enregistrement ou d'une ou plusieurs cellules
dans la table « T_A », dans la colonne « C_A » de la famille de colonnes
« CF_A » à la clé « ROW_A », il faut exécuter la commande suivante :
          hbase> get 'T_A', 'ROW_A', 'CF_A:C_A'
Il est à noter qu'il est possible de lire une version spécifique d'une
colonne en spécifiant le numéro de la version.
```

1.2.4.1 L'opération de suppression « DELETE »

L'instruction qui permet de supprimer le contenu d'un enregistrement, d'une famille de colonnes, d'une version d'une colonne ou des toutes les versions d'une colonne est celle appelé « DELETE ».

1.2.4.2 L'opération de recherche « SCAN »

La dernière opération et la plus importantes est celle de la recherche. Cette opération permet de filtrer les données par les valeurs de clé, les noms de familles de colonnes, les nomes de colonnes, les valeurs elles-mêmes et l'estampille temporelle « timestamp » des enregistrements. Comme l'enregistrement sont classé en fonction de leurs clé, il est entendu que la recherche la plus efficace est par la clé et non par la valeur d'une cellule.

1.2.5 Console de HBase

La console de HBase est base sur le module interactif de Ruby (IRB – Interactive Ruby Shell). Cette console permet d'exécuter du code JRuby. Qu'est que JRuby ? C'est le langage Ruby mais qui a été implémenté en Java car la version original de Ruby est implémenté en C. IRB est un module formidable de Ruby qui permet d'écrire et d'exécuter du code à partir de la console de HBase, ce qui permet d'interroger HBase avec des commandes complexes directement à partir de la console de HBase. C'est simple, efficace et rapide. (Programming Ruby: The Pragmatic Programmer's Guide)

1.2.6 Quand utiliser HBase ?

Il est recommandé d'utiliser HBase seulement pour l'entreposage et accès en écriture/lecture à une très grande quantité de données. Ce qui est sous-entendu par une très grande quantité de données, c'est des données de plusieurs billions d'enregistrements à l'intérieur d'une seule table.

1.3 Distribution commercialisé de Hadoop : Cloudera

Cloudera est une distribution stable de l'écosystème Hadoop d'Apache. Elle vise à faciliter l'installation de HDFS, Hive, Sqoop, ZooKeeper, HBase et plusieurs autres composantes de Hadoop dans un environnement commercial ou non commercial. Sans la distribution de Cloudera un déploiement de Hadoop avec toutes les différentes versions de ses composantes, prenait énormément de temps et en plus il était très difficile de le faire. Ce problème n'existe plus grâce à la solution qui est disponible gracieusement par Cloudera[1].

1.4 Base de données relationnelles orienté objet : PostgreSQL

PostgreSQL est une base de donnée « open source », orienté objet et relationnel. Elle offre toutes les mêmes fonctionnalités qu'une base de données commercialisée tel qu'Oracle ou SQL Server. Fait très intéressant, elle peut être utilisée sur tous les systèmes d'exploitation. De plus elle peut être utilisé par les applications développée dans n'importe quel langage, tel que : C/C++, Java, .Net et plusieurs d'autres. Par contre comme toute base de données relationnelle elle est limitée au niveau de grandeurs de ses tables à 32 TB. [2]

1.5 Sqoop : Utilitaire de transfert de données

Sqoop « SQL-to-Hadoop » est un outil développé par l'équipe d'Apache dans le but de transférer les données en vrac « bulk » entre HDFS, Hive ou HBase et de base de données relationnelles. Le fait très intéressant est que la plupart de développeurs travaillant sur ce

---

[1] (Cloudera's Distribution details)

[2] (PostgreSQL : About)

projet sont des employés de Cloudera. Aussi il est à noter que cet outil fait partie de la distribution de Hadoop par Cloudera (CDH).

Avant l'existence de Sqoop, tout le processus d'import/d'export des données entre HDFS et de système de gestion de base de données relationnelle (RDBMS) était fait en utilisant MapReduce. Grâce à Sqoop tout ce processus lourd et complexe a était automatisé et est désormais disponible via quelques lignes de commandes.

CHAPITRE 2

ENVIRONNEMENT DE DEVELOPPEMENT CLOUDERA SOUS LINUX

2.1 Installation de Hadoop

La distribution de Hadoop par Cloudera (CDH3) est disponible sur le site de Cloudera dans différents formats et pour différents systèmes d'exploitation. Dans le but d'étude et de familiarisation avec l'environnement de Hadoop, la machine virtuelle pour VMWare [3]est un très bon choix par contre c'est un environnement très lent et ne peut pas être utilisé pour faire du développement. C'est pour cette raison que l'environnement de développement est composé d'un serveur Ubuntu hébergé sur le réseau de l'ÉTS et accessible par le développeur à travers VPN et SSH à partir de son portable. Toutes les instructions d'installations sont documentées et disponibles sur le site web de Cloudera[4].

L'annexe I contient toutes les étapes de configuration de l'environnement de développement Hadoop pour un serveur Ubuntu.

---

[3] https://downloads.cloudera.com/demo_vm/vmware/cloudera-demo-vm-cdh3u3-vmware.tar.gz

[4] https://ccp.cloudera.com/display/CDHDOC/CDH3+Installation#CDH3Installation-InstallingCDH3onUbuntuandDebianSystems

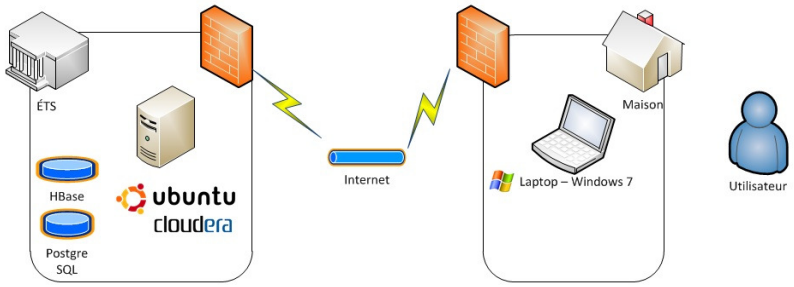


Figure 4 : Schéma de l'environnement de développement Cloudera sous Linux

2.2 Configuration de HBase

Notre configuration de HBase est celle par défaut de HBase. Nous avons choisi ce mode pour les raisons suivant :

1) Nous disposons d'un seul serveur, ce qui nous enlève automatiquement la possibilité de faire une installation entièrement distribué. Car cette dernière exige au moins deux serveurs.
2) Nous avons écarté l'option d'une installation pseudo-distribué car la quantité des données n'était pas suffisamment grande pour utiliser HDFS. Le système local de gestion de fichier répond entièrement à nos besoins qui sont juste au niveau de familiarisation avec HBase.

2.3 Installation et « restore » de la base de donnée PostgreSQL

Nous avons choisi d'installer PostgreSQL version 9.0 sur le poste du développeur ainsi que sur le serveur Ubuntu. La base de données sur le poste du développeur est utilisée pour analyser le schéma existant ainsi que pour analyser les requêtes problématiques. L'installation sur le serveur Ubuntu est par contre utilisée pour le transfert de données entre PostgreSQl et HBase.

2.4 XMing – Accès à distance

Nous avons choisi d'utiliser Xming pour être capable de se connecter à distance sur le serveur Ubuntu. Cet outil offre les mêmes fonctionnalités que « Remote Desktop » de Microsoft. Veuillez consulter l'ANNEXE V pour les instructions de connexion vers le serveur Ubuntu.

2.5 FileZilla – Échange de fichiers

Nous avons choisi d'installer l'utilitaire gratuit « FileZilla » pour faciliter l'échange de fichiers entre le serveur UBuntu et le portable du développeur. C'est un outil qui permet de transférer de façon sécuritaire des très gros fichiers. Particulièrement il faut être en mesure de transférer tous les scripts SQL reçus par Abraham II Gómez Morales, à partir du poste de développeur vers le serveur qui se trouve sur le réseau de l'université (51 Go des données).

CHAPITRE 3

MIGRATION DE POSTGRESQL VERS HBASE

3.1 Introduction

La base de données PostgreSQL est une base de données relationnelle qui contient un échantillon des données de la base de données MySQL du Centre Hospitalier de l'Université de Montréal (CRCHUM). Les données entreposées via un système de *séquençage génomique* des patients produisent des quantités gigantesques de données à tous les jours. Ces données sont des données historiques et sont utilisés en consultation dans le but d'identifier des liens entre la génomique de patients atteint d'une maladie et ceux qui ne le sont pas. Puisque les tables qui contiennent les données historiques sont très volumineuses, les requêtes exécutées contre ces tables deviennent de plus en plus lentes.

Dans les sections suivantes nous allons présenter le schéma actuel de PostgreSQL, les requêtes problématiques, l'objectif de la migration de PostgreSQL vers HBase ainsi qu'une comparaison des résultats obtenu en interrogeant les deux bases de données.

3.2 Schéma Actuel de PostgreSQL

L'analyse des requêtes problématique ainsi que l'analyse du schéma de PostgreSQL nous a permis de reconstruire le modèle de données du schéma actuelle PostgreSQL. Ce schéma se compose de 13 tables et peut être regroupé en trois sous-schémas décrits dans les paragraphes suivants.

Les tables de références pour décrire les algorithmes de séquençage ainsi que leurs relations avec la table de pipeline de dépistage. Toutes ces tables sont très peu volumineuses et par conséquent non candidates pour la migration vers HBase dans le cadre de cette étude.

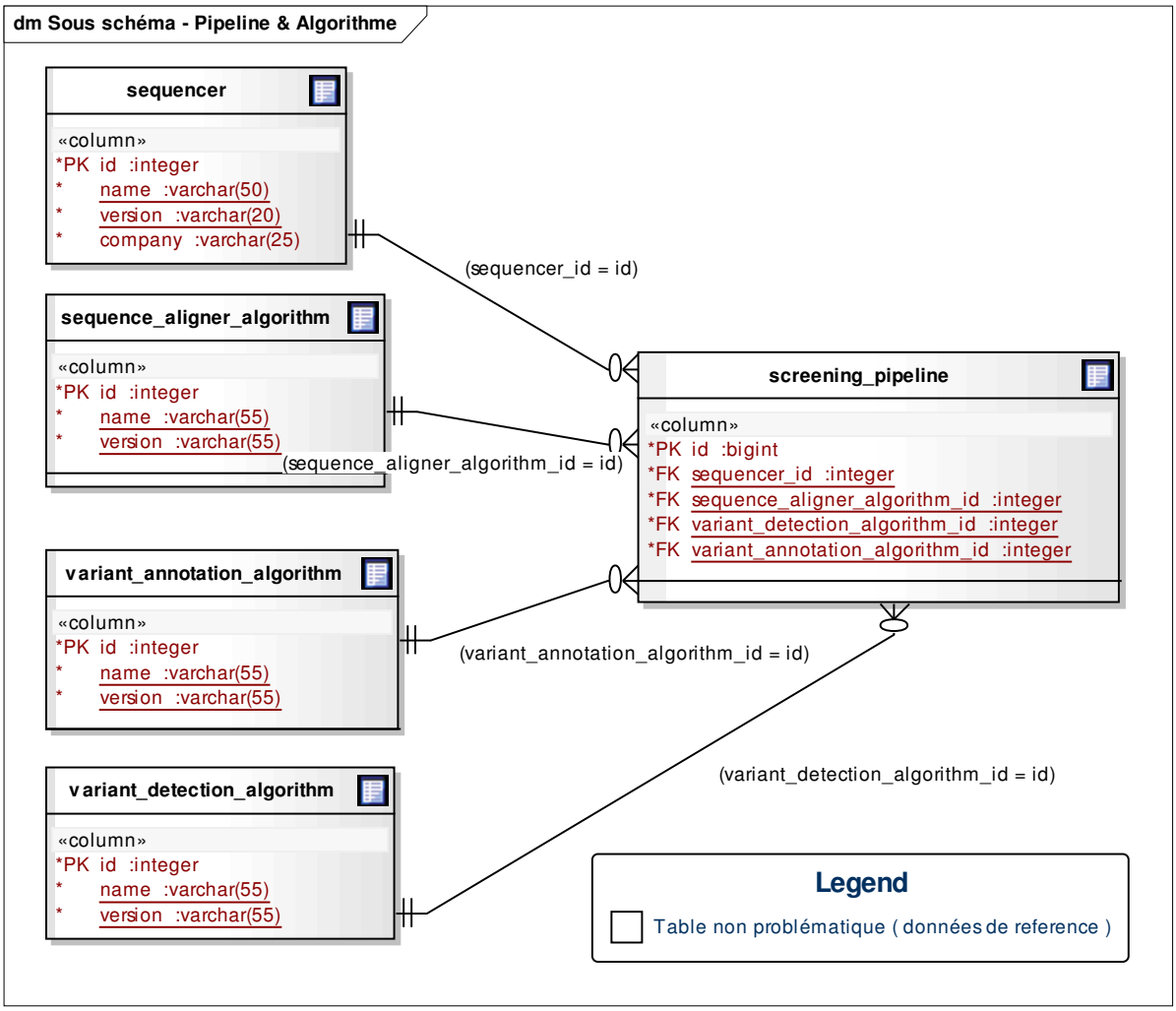


Figure 5 : Sous-schéma – Pipeline et Algorithme - PostgreSQL

Les tables de références pour décrire les génomes humains 18 et 19 ainsi que leurs relations avec la table de chromosome et la table de positions. Les tables « ngs_hg19_postion » et « ngs_hg18_postion » sont très volumineuses par contre non référencées par les requêtes problématiques. Par conséquent elles ne sont candidates pour la migration vers HBase dans le cadre de cette étude.

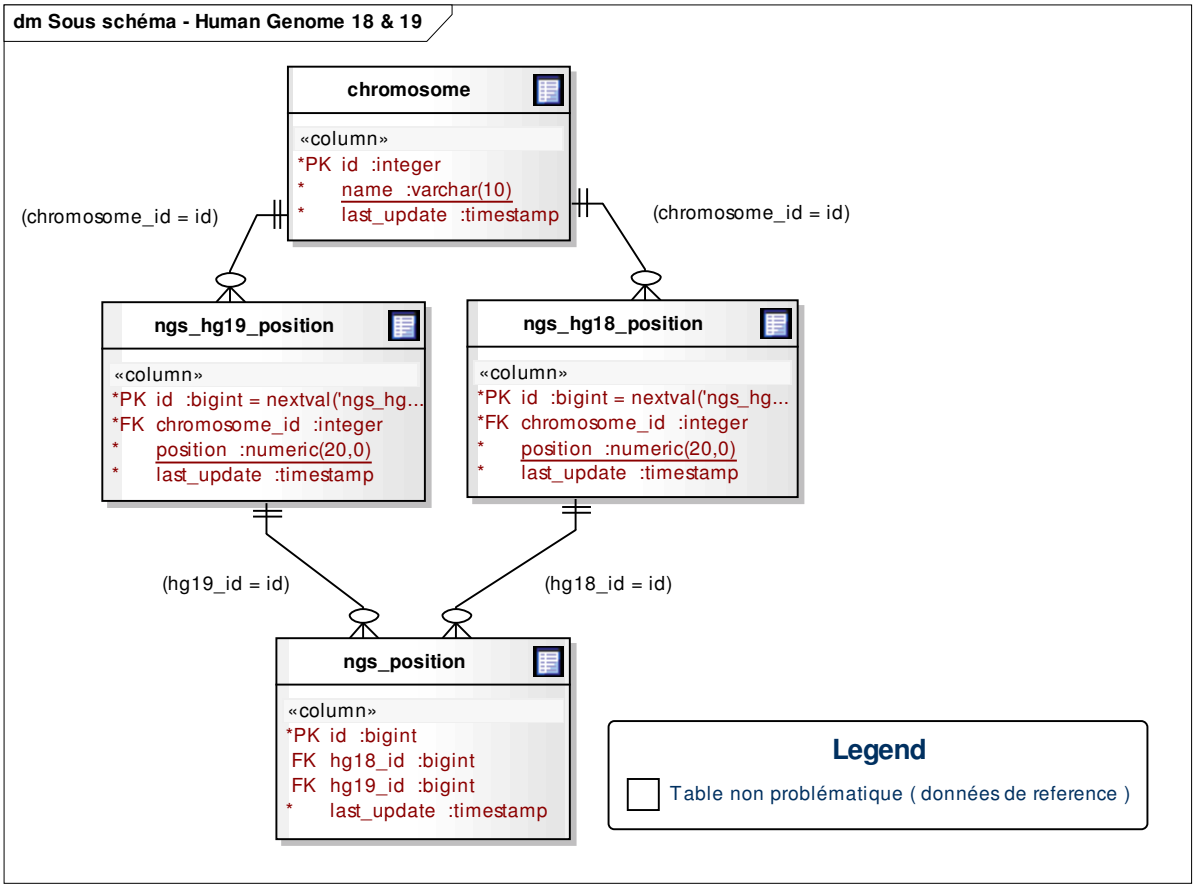


Figure 6 : Sous-schéma - HG 18 et 19 - PostgreSQL

Toutes les tables qui contient les données d'échantillons (d'individus) ainsi que leurs variant selon le chromosome ou mutation sont extrêmement volumineuses (plus de 50 000 000 d'enregistrements par table). De plus ce sont des données historiques qui ne sont pas sujet à une mise à jour. Par conséquente elles sont toutes identifié comme problématique et par ce fait nous le migrons vers HBase.

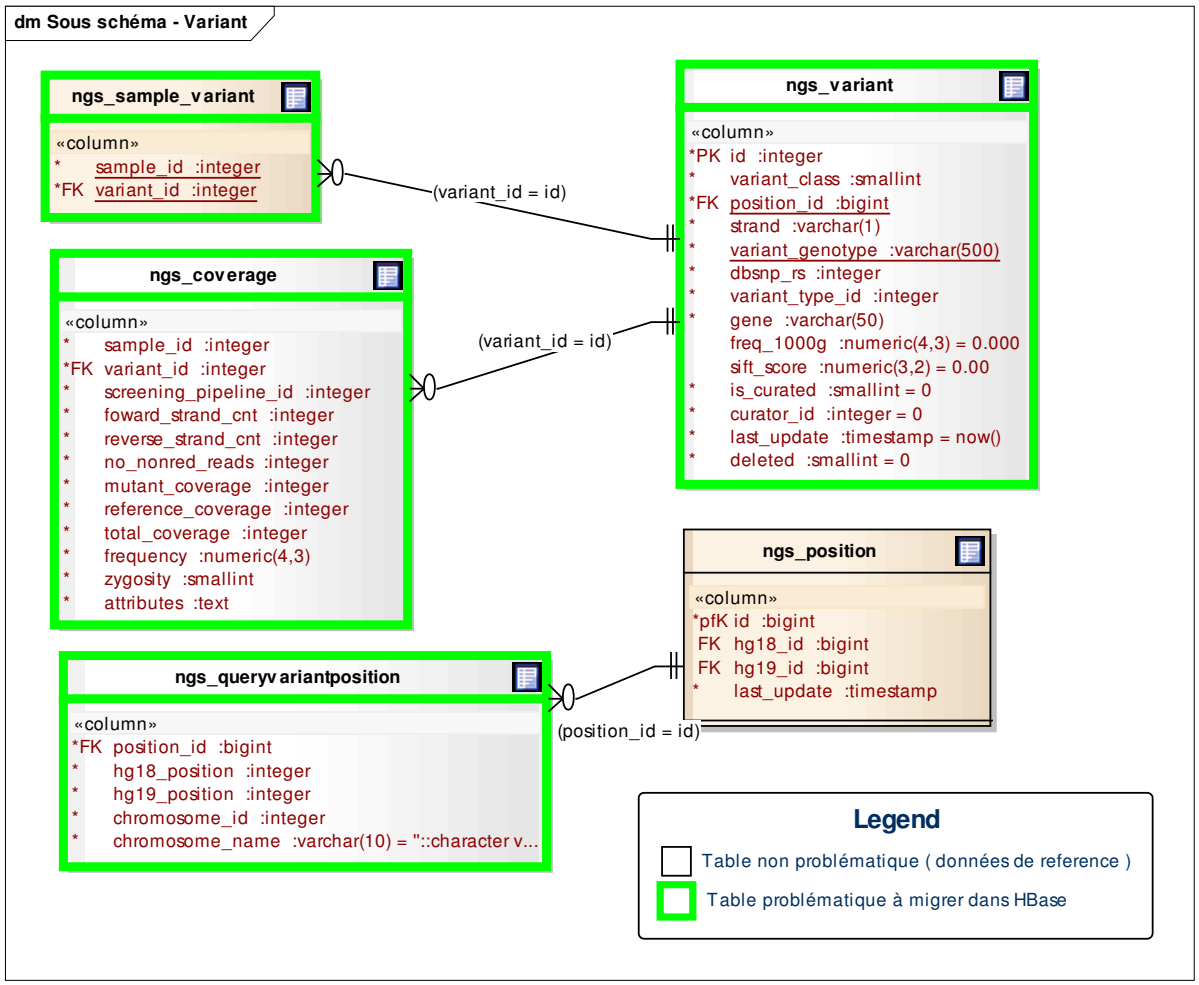


Figure 7 : Sous-schéma – Variant & Coverage – PostgreSQL

3.3 Analyse de requêtes problématiques

Les requêtes les plus problématiques sont celles qui cherchent à identifier les « Variant » qui sont associé à un sous-ensemble d'échantillons et non associé à un autre sous-ensemble d'échantillons selon les besoins de l'utilisateur qui interroge le système.
L'usager a comme première option d'exclure les variants présents dans certains échantillons « sample » (individus). Pour obtenir cette liste de variants à exclure du rapport, la requête suivante est utilisée :

Tableau 4 : 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"
```

Un deuxième choix s'offre à l'usager où il peut non seulement inclure des variants présents dans certains échantillons « sample » (individus) mais aussi qui ont était séquencé en utilisant un ou plusieurs pipelines de dépistage spécifiques. Pour obtenir cette liste de variants à inclure dans le rapport, la requête suivante est utilisée :

Tableau 5 : Requête # 2.a - 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"
```

Finalement un troisième choix s'offre à l'usager qui veut inclure tous les variants, peu importe pipeline de dépistage y associé. Pour obtenir cette liste de variants à inclure dans le rapport, la requête suivante est utilisée :

Tableau 6 : Requête # 2.b - Variant à inclure dans le résultat de recherche pour n'importe lequel 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"
```

Finalement la requête problématique identifiée par le responsable IT du CRCHUM, est celle qui tente de sortir toutes les variant en mutation « missense variants » présents dans la famille A mais qui ne sont pas présents dans le control B et qui sont associés au Pipeline de dépistage C. Pour afficher ce rapport à l'usager les requêtes suivantes doivent être exécutées : requête # 1, requête # 2.a et requête # 3.

Tableau 7 : Requête # 3 - Requête problématique

```
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
```

3.4 Objectif de la migration de PostgreSQL vers HBase

Le premier objectif de notre migration des données de PostgreSQL vers HBase est de bâtir une base de données HBase sous Hadoop avec des données réelles pour se familiariser avec le model de données d'une base de donnée non-relationnelle ainsi que les opérations qui sont supporté par une telle base de donnée. Le deuxième objectif de notre migration est d'être capable d'exécuter toutes les requêtes problématiques de la section 3.3 sur HBase. Pour par

la suite les comparer avec les résultats des requêtes exécutés sur PostgreSQL au niveau du temps de réponses et la validité de données. Tout ceci dans le but de vérifier si un model non-relationnel répond aussi bien au besoin de CRCHUM qu'un model relationnel. Le dernier objectif de notre migration ce qu'il doit être possible de reproduire cette migration autant de fois qu'nous le désirons dans un laps de temps de moins de quelques heures et avec de bases de données d'un volume considérable (plus de 50 millions d'enregistrement).

3.5 Description de la migration de PostgreSQL vers HBase

Après une étude détaillée du schéma actuel de PostgreSQL et l'analyse de requêtes problématiques, nous avons très vite constaté que nous avons des millions d'enregistrement à migrer vers HBase. De plus cette migration doit s'effectuer dans un délai très court dû à nos contraintes de temps. C'est avec cette contrainte comme critère que nous avons fait plusieurs recherches pour être finalement dirigé vers un outil incroyable de Hadoop qui porte le nom de Sqoop. Selon les spécifications décrites sur le site web de Sqoop[5] et l'aide de Patrice Dion le transfert de nos données peut s'effectue via Sqoop via une ligne de commande et à partir de n'importe quelle requettes SQL. Une fois les données transférées, il nous doit être possible d'interoger HBase pour obtnir les mêmes informations qu'à partir de PostgreSQL. Ce dernier critère nous oblige à denormaliser les données à transferer d'une façon où en utilisant les options des opérations de recherche de HBase l'information pourra être retrouver très facilement.

Avant de débuter la migration, nous avons effecté l'installation de PostgreSQL ainsi que de l'outilitaire Sqoop sur le serveur Ubuntu. Nous avons opté pour deplacer PostgreSQL de notre machine cliente (qui ne se trouve pas sur le reseau de l'ETS) vers le serveur Ubuntu (qui contient Hadoop et HBase) pour éviter tous les problèmes de securité et pour maximiser la vitesse du transfert.

---

[5] http://nosql.mypopescu.com/post/1541593207/quick-reference-hadoop-tools-ecosystem

Une fois que les données ont été migrées vers HBase, toutes les requêtes ont été effectuées via le Shell de HBase avec le langage de script, orienté-objet JRuby.

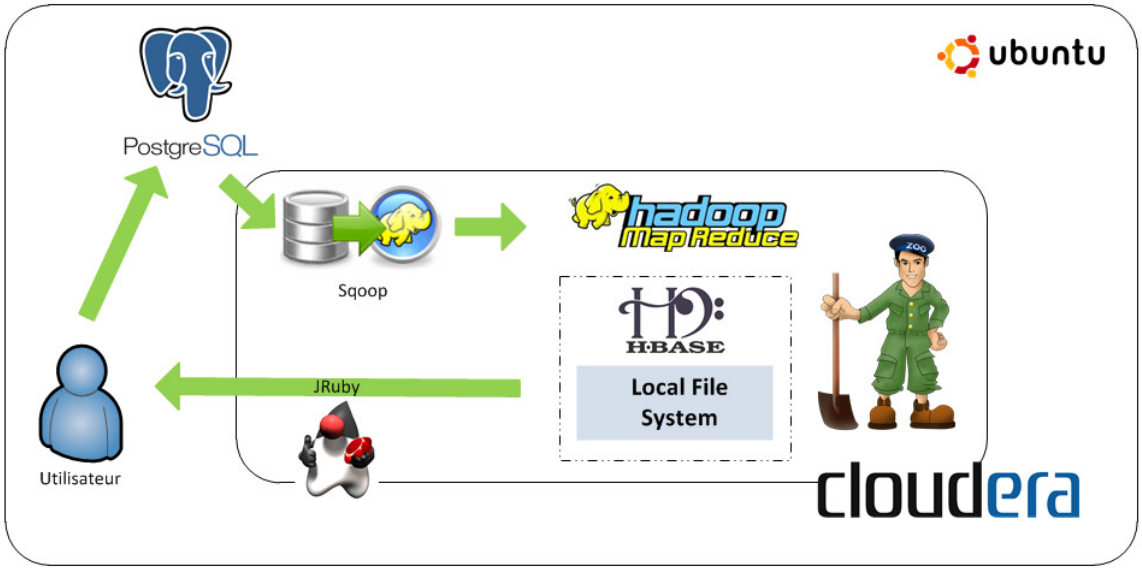


Figure 8 : Migration de PostgreSQL vers HBase

3.5.1 Dénormalisation des données et leurs transfert de PostgreSQL vers HBase

Les données sujet à migration sont toutes celles utilisées par les requêtes de la section 3.3 et identifié plus précisément dans le model de données (voir Figure 7 : Sous-schéma – Variant & Coverage – PostgreSQL).

Premièrement pour chaque requête de la section 3.3, nous avons créé une vue dans PostgreSQL. Nous avons aussi ajouté une colonne calculée qui contient la nouvelle clé pour chaque enregistrement et qui servira de clé primaire une fois importé dans HBase. La nouvel clé doit contenir toute l'information nécessaire pour la recherche. Par exemple dans la requête # 1, le système doit être capable de trouvé tous les valeurs de « variant_id » qui correspond à un échantillon donnée. Par conséquent la clé doit contenir la valeur du échantillon et du variant.

Tableau 8 : Script de création de vues

```
CREATE OR REPLACE VIEW "NGS_query_1_ou_2b_view" AS
 SELECT
   (to_char(sample_id,'000000') || ':' || to_char(variant_id,'000000000000000')) as v_key,
        sample_id, variant_id
   FROM ngs_sample_variant;


CREATE OR REPLACE VIEW "NGS_query_2a_view" AS
 SELECT (to_char(screening_pipeline_id,'000000') || ':' ||
        to_char(sample_id,'000000') || ':' ||
        to_char(variant_id,'000000000000000')) as v_key,
        ngc.variant_id
   FROM ngs_coverage ngc;


CREATE OR REPLACE VIEW "NGS_query_3_view" AS
 SELECT (to_char(v.variant_type_id,'000000') || ':' ||
        to_char(v.id,'000000000000000')) AS v_key,
        v.variant_type_id,
        v.id as variant_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
  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;
```

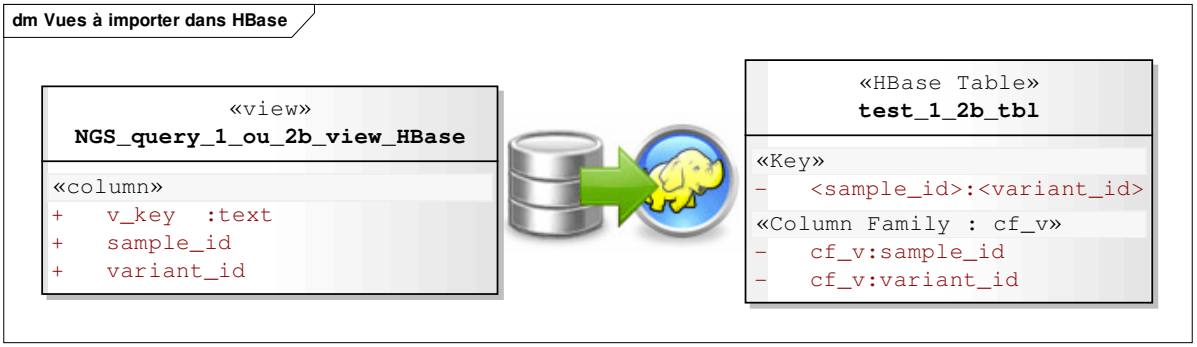


Tableau 9 : Schéma HBase de la table « test_3_tbl »

Nous avons transféré tous les données de toutes les colonnes de la table « ngs_query_1_ou_2b_view_HBase » vers HBase dans une seule famille de colonnes nommé « cf_v ». Ce transfert a été effectué avec la commande « import » de Sqoop avec les paramètres suivants et a pris 27 min pour 54 695 435 enregistrements :

« --query » : la requête SQL qui retourne les données à transférer;

« --hbase-table » : le nom de la table de destination (existante ou nouvelle);

« --columns » : toutes les colonnes à transférer (très important de toujours inclure la colonne de la clé primaire);

« --column-family » : le nom de la famille de colonnes de destination (pour un transfert vers plus qu'une famille de colonnes, il faut créer un script par famille de colonnes car il n'est pas possible de spécifier plus qu'un nom par commande);

« --hbase-row-key » : le nom de la colonne qui contient la clé primaire;

« --hbase-create-table » : dans le cas où la table de destination n'existe pas, Sqoop s'occupe de la crée;

« --split-by » : indique par quelle colonne les données doivent être indexé (clé primaire), dans le cas où plus qu'un enregistrement contient la même clé, Sqoop enregistre les valeurs dans le même enregistrement mais avec une version ultérieur;

« --num-mappers » : indique combien le nombre de tâche de MapReduce à exécuter en parallèle.

Tableau 10 : Script de transfert de PostgreSQL vers HBase via Sqoop

```
sqoop import --connect jdbc:postgresql://localhost:5432/20120224-BDS2D-
New_Schema --username postgres --password postgres
```

```
--query “select v_key,sample_id,variant_id from \”NGS_query_1_ou_2b_view\”
where \$CONDITIONS” --hbase-table test_1_2b_tbl
--columns v_key,sample_id,variant_id
--column-family cf_v --hbase-row-key v_key --hbase-create-table -split-by
v_key -num-mappers 16
```

Temps : 1,673 sec = 27 min, Map output records : 54 695 435.

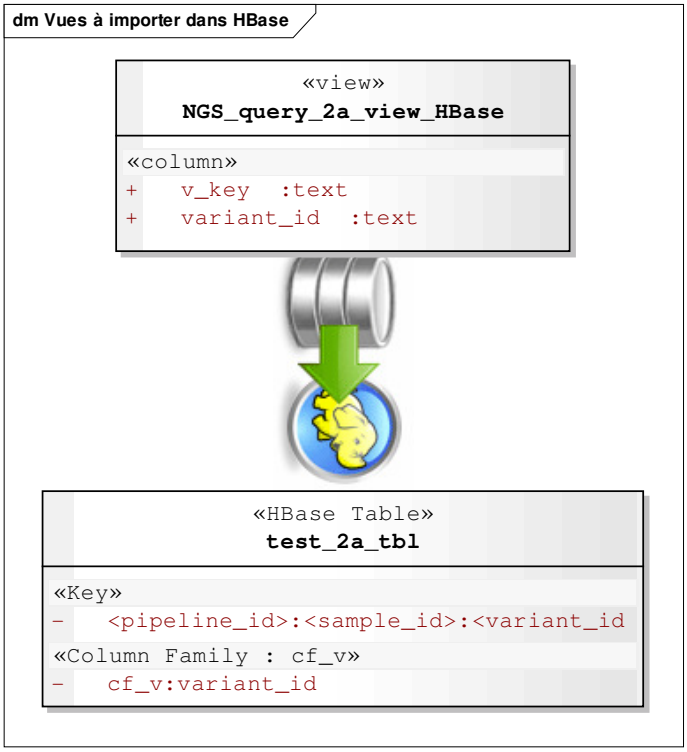


Tableau 11 : Schéma HBase de la table « test_3_tbl »

Nous avons transféré tous les données de toutes les colonnes de la table « ngs_query_2a_view_HBase » vers HBase dans une seule famille de colonnes nommé « cf_v ». Ce transfert a été effectué avec la commande « import » de Sqoop avec les paramètres suivants et a pris 35.3 min pour 56 571 745 enregistrements.

Tableau 12 : Script de transfert de PostgreSQL vers HBase via Sqoop

```
sqoop import --connect jdbc:postgresql://localhost:5432/20120224-BDS2D-
New_Schema --username postgres --password postgres
--query "select v_key, variant_id from \"NGS_query_2a_view\" where
\$CONDITIONS" --hbase-table test_2a_tbl
--columns v_key, variant_id
--column-family cf_v --hbase-row-key v_key --hbase-create-table -split-by
v_key


Temps : 2,118 sec = 35.3 min, Map output records : 56 571 745.
```

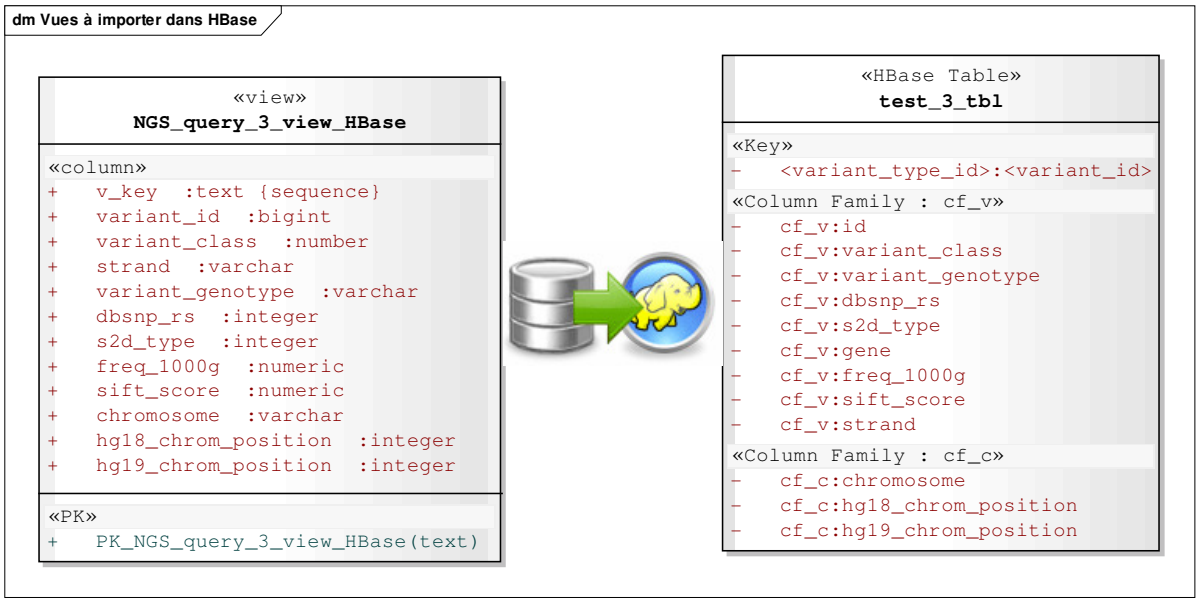


Tableau 13 : Schéma HBase de la table « test_3_tbl »

Nous avons transféré la table « ngs_query_3_view_HBase » vers HBase dans deux familles de colonnes nommées « cf_v » et « cf_c ». Ce transfert a été effectué avec deux commandes « import » de Sqoop avec les paramètres suivants. Le premier transfert de neuf colonnes a pris 55 min pour 26 975 977 enregistrements et le deuxième transfert dans la même vue mais dans une famille de colonnes différentes, de trois colonnes, a pris 20 min pour le même nombre d'enregistrements.

Tableau 14 : Script de transfert de PostgreSQL vers HBase via Sqoop

```
sqoop import --connect jdbc:postgresql://localhost:5432/20120224-BDS2D-
New_Schema --username postgres --password postgres
--query "select v_key,variant_type_id,variant_id,variant_class,strand,
variant_genotype,dbsnp_rs,s2d_type,gene,freq_1000g,sift_score
 from \"NGS_query_3_view\" where \$CONDITIONS" --hbase-table test_3_tbl
--columns v_key,variant_type_id,variant_id,variant_class,strand,
variant_genotype,dbsnp_rs,s2d_type,gene,freq_1000g,sift_score
--column-family cf_v --hbase-row-key v_key --hbase-create-table -split-by
v_key
Temps : 3,298 sec = 55 min, Map output records : 26 975 977.

sqoop import --connect jdbc:postgresql://localhost:5432/20120224-BDS2D-
New_Schema --username postgres --password postgres
--query "select v_key,chromosome,hg18_chrom_position,hg19_chrom_position
 from \"NGS_query_3_view\" where \$CONDITIONS" --hbase-table test_3_tbl
--columns v_key,chromosome,hg18_chrom_position,hg19_chrom_position
--column-family cf_c --hbase-row-key v_key --hbase-create-table -split-by
v_key
Temps : 1,227 sec = 20.45 min, Map output records : 26 975 977.
```

3.5.2 Traduction des requêtes problématiques SQL vers HiveQL

Suite à plusieurs analyses et essaies, il nous a été impossible de traduire les requêtes 1, 2a et 2b en requêtes HBase et obtenir le même résultat. Le gros problème avec HBase c'est qu'il ne supporte pas de requêtes avec la clause « Select DISTINCT … ». Par contre il nous a été possible de traduire la requête # 3 en code JRuby.

Les prochains tableaux illustrent un exemple de requête SQL que nous avons traduit en commandes SCAN et GET. Ceci dans le but de comparer le temps de réponse d'une requête SQL exécuté dans PostgreSQL et des ces commandes équivalente exécutés dans la console de HBase. Il ne faut pas considérer ces données comme étant très significatives à cause de la configuration de notre environnement de développement. C'est très important de prendre en considération que le mode d'exécution de HBase est celui par défaut qui est non distribué. Comme mentionnée dans le premier chapitre un tel mode d'exécution ne devrait surtout pas être utilisé pour de tests de performance. Par contre ces résultats nous démontrent qu'on peut traduire une requête SQL en commande HBase et obtenir le même résultat dans un temps de réponse très semblable.

Tableau 15 : PostgreSQL - Requête SQL # 3

```
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 ( 53,54,55,56,57,58,59,60,61 )
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
```

Tableau 16 : HBase – Requête # 3 traduite avec la commande « SCAN : start/end index»

```
scan 'test_3_tbl', { COLUMNS => 'cf_v:variant_id', LIMIT => 5, STARTROW => '
000004: 000000000000053', STOPROW => ' 000004: 000000000000062'}
```

Tableau 17 : HBase – Requête # 3 traduite avec plusieurs commandes « GET »

```
get 'test_3_tbl', ' 000004: 000000000000053', { COLUMNS => 'cf_v:variant_id'}
get 'test_3_tbl', ' 000004: 000000000000054', { COLUMNS => 'cf_v:variant_id'}
get 'test_3_tbl', ' 000004: 000000000000055', { COLUMNS => 'cf_v:variant_id'}
get 'test_3_tbl', ' 000004: 000000000000056', { COLUMNS => 'cf_v:variant_id'}
get 'test_3_tbl', ' 000004: 000000000000057', { COLUMNS => 'cf_v:variant_id'}
get 'test_3_tbl', ' 000004: 000000000000058', { COLUMNS => 'cf_v:variant_id'}
get 'test_3_tbl', ' 000004: 000000000000059', { COLUMNS => 'cf_v:variant_id'}
get 'test_3_tbl', ' 000004: 000000000000060', { COLUMNS => 'cf_v:variant_id'}
get 'test_3_tbl', ' 000004: 000000000000061', { COLUMNS => 'cf_v:variant_id'}
```

Tableau 18 : Comparaison de temps de réponse d'une requête SQL et d'une requête NoSQL

Essaie	Temps d'exécution en seconde (s)		
	PostgreSQL	HBase	
	Tableau 15 : PostgreSQL - Requête SQL # 3	Tableau 16 : HBase – Requête # 3 traduite avec la commande « SCAN : start/end index»	Tableau 17 : HBase – Requête # 3 traduite avec plusieurs commandes « GET »
Connexion # 1			
1	0.368 s	0.3610 s	0.261
2	0.100 s	0.0670 s	0.133
Connexion # 2			
1	0.243 s	0.3680	0.126 (.018+.016+.011+.01+.012+.009+.011+.024+.015
2	0.100 s	0.0250	0.105 (.021+.012+.011+.01+.012+.009+.012+.009+.009)
3	0.248 s	0.0270	0.097 (.015+.011+.009+.008+.009+.009+.016+.009+.011)
4	0.100 s	0.0350	0.101 (.013+.022+.01+.009+.009+.009+.01+.009+.01)
5	0.200 s	0.0220	0.14 (.012+.011+.01+.017+.013+.013+.012+.018+.034)

CONCLUSION

Depuis l'apparition de Facebook, Amazon et Google, l'informatique de nuage prend de plus en plus de place dans la société d'aujourd'hui par contre peu connaissent les technologies qui sont derrière ces noms. Comme c'est un sujet de l'heure et que tout le monde en parle, l'initiative de ce projet était de combler ce manque de connaissance et de créer un environnement avec quelques projets qui touche à ces technologies.

Ce projet visait la création d'un environnement « type » sous Hadoop avec Cloudera dans le but de le réutiliser pour le nouveau cours de M. Alain April au sujet de « Cloud Computing » et « Hadoop ». De plus il visait l'utilisation d'un sous-ensemble de données (problématiques) d'une étude génomique de CRCHUM dans le but d'étudier la faisabilité de traduction des requêtes SQL vers des commandes HBase.

L'échantillon des données de la génomique des patients du Centre Hospitalier de l'Université de Montréal (CRCHUM) a permis de valider la migration de la base de données relationnelle PostgreSQL vers la base de données non-relationnelle HBase. La traduction des requêtes problématiques a permis d'identifier les avantages et les limites de HBase.

L'environnement de Hadoop de la distribution Cloudera sous Ubuntu, a permis de valider que les nouvelles versions de projets de l'écosystème Hadoop, sont de plus en plus stables et facile à déployer. Aussi grâce à l'utilitaire Sqoop, une migration d'une base de données relationnelle vers une base de données non-relationnelle est devenue très facile et rapide. De plus la prévue de concept en JRuby nous a permis de traduire les requêtes SQL et démontrer les limites ainsi que la puissance de HBase. Suite à la tentative de traduction des requêtes contenant la clause « DISTINCT » cette étude a permis de découvrir les limites de HBase par rapport au relations de type « Plusieurs à Plusieurs ».

Selon nous, une première version d'un modèle de schéma de la base de données HBase a été définit par contre il n'est pas parfait et devrait être revue et adapté pour permettre de faire de recherche sur les données qui contient des relations de type « Plusieurs à Plusieurs »

RECOMMANDATIONS

Lors de la traduction de requêtes SQL en requêtes NoSQL et la connexion vers la base de donnée HBase à partir d'un poste client, nous n'avons pas pu se connecter à HBase qui est situé sur le serveur Ubuntu. Sûrement ce problème est relier à la configuration de la sécurité du serveur Ubuntu et surement la complexité de configuration de sécurité de HBase. Ce point reste à investiguer dans un projet futur.

HBase est définitivement très efficace en écriture par contre, très peu flexible en mode lecture et recherche. Le seul moyen efficace d'accéder aux données et via l'information qui est contenu dans l'index (clé) de chaque enregistrement ou bien dans le nom d'une colonne ou d'une famille de colonnes. Toute recherche de donnée via d'autre information que celles incluses dans l'index et à éviter car beaucoup moins performante qu'une recherche dans une base de donnée relationnelle.

Pour la suite de ce travail, nous suggérons d'analyser toutes les requêtes problématiques du centre Hospitalier de l'Université de Montréal (CRCHUM). Ce travail permettrait de revoir s'il ne serait pas possible de créer de transformations « MapReduce » pour chaque type de requête. Par la suite lors de lancement d'une requête par l'usager le système déciderais quel table HBase il doit interroger et obtenir l'information que via une recherche par l'index.

BIBLIOGRAPHIE

CHAPITRE 4BIBLIOGRAPHIE

(s.d.). Consulté le Mars 11, 2012, sur Hbase/Shell - Hadoop Wiki: http://wiki.apache.org/hadoop/Hbase/Shell

HDFS Architecture Guide . (2012). Consulté le 03 05, 2012, sur Hadoop: http://hadoop.apache.org/common/docs/current/hdfs_design.html

Apache Sqoop. (s.d.). Consulté le Avril 07, 2012, sur http://sqoop.apache.org/

Cloudera - Apache Sqoop – Overview. (s.d.). Consulté le Mars 19, 2012, sur www.cloudera.com: http://www.cloudera.com/blog/2011/10/apache-sqoop-overview/

Cloudera's Distribution details. (s.d.). Consulté le Janvier 2012, sur www.cloudera.com: http://www.cloudera.com/hadoop/

Foundation., C. ©. (2011). *Welcome to Apache™ Hadoop™!* Consulté le Janvier 2012, sur Hadoop: http://hadoop.apache.org/#What+Is+Apache+Hadoop%3F

George, L. (2011). *HBase : The Definitive Guide.* O'REILLY.

HBase:Architecture » New IT Farmer. (s.d.). Consulté le Avril 6, 2012, sur http://blog.newitfarmer.com/nosql/hbase-nosql/4385/repost-hbasearchitecture/

Lars, G. (2011). *HBase : The Definitive Guide.* O'REILLY.

Popescu, A. (s.d.). *Bulk Importing Data Into HBase*. Consulté le Mars 19, 2012, sur http://nosql.mypopescu.com/post/9034681586/bulk-importing-data-into-hbase

PostgreSQL : About. (s.d.). Consulté le Mars 12, 2012, sur www.postgresql.org: http://www.postgresql.org/about

Programming Ruby: The Pragmatic Programmer's Guide. (s.d.). Consulté le Avril 6, 2012, sur Ruby Central Inc.: http://www.rubycentral.com/pickaxe/irb.html

White, T. (2010). *Hadoop : The Definitive Guide.* O'REILLY.

ANNEXE I

INSTALLATION ET CONFIGURATION DE L'ENVIRONNEMENT DE DÉVELOPPEMENT HADOOP SUR UN SERVEUR UBUNTU

Installation de CDH 3 sur un serveur Ubuntu[6]

Étape 1 : Télécharger le répertoire CDH3 ou le « package » CDH3

Note : l'adresse IP du serveur Ubuntu : 10.194.32.172

Pour télécharger et installer le packege CDH3 :

1. Cliquez sur le lien suivant CDH3
2. Pour installer le « package » exécuter la commande suivante :

```
sudo dpkg -i Downloads/cdh3-repository_1.0_all.deb
```

Création du répertoire CDH3 :

1. Créer le fichier « `/etc/apt/sources.list.d/cloudera.list` » avec le contenu suivant :

```
deb http://archive.cloudera.com/debian lucid-cdh3 contrib
deb-src http://archive.cloudera.com/debian lucid-cdh3 contrib
```

Étape 2 : Installation de CDG3 :

1. Metter à jout l'index du package APT:

```
$ sudo apt-get update
```

2. Installer Hadoop.

```
$ apt-cache search hadoop
$ sudo apt-get install hadoop-0.20
```

3. Configurer le chemin de Java dans le fichier de configuration de hadoop:
    1. Ouvrir le fichier "/etc/hadoop/conf/hadoop-env.sh"
    2. Modifier la valeur de la variable « JAVA_HOME = /usr/lib/jvm/java-6-sun »

[6] https://ccp.cloudera.com/display/CDHDOC/CDH3+Installation#CDH3Installation-InstallingCDH3onUbuntuandDebianSystems

3. Fermer et sauvegarder le fichier

2. Modifier la variable globale JAVA_HOME avec la commande suivante :
   export JAVA_HOME=/usr/lib/jvm/java-6-sun

Déploiement de CDH3 en mode autonome « standalone »

Déploiement en mode autonome de CDH3:

1. Vérifier que le gestionnaire de package et capable de lister tous les packages de Hadoop

```
$ apt-cache search hadoop
```

2. Installer les packages de Hadoop.

```
$ sudo apt-get install hadoop-0.20
```

3. Lister les fichiers installés

```
$ dpkg -L hadoop-0.20
```

Les répertoires suivants devraient maintenant exister sur votre serveur :

```
Hadoop wrapper script          /usr/bin/hadoop-0.20
Hadoop Configuration Files     /etc/hadoop-0.20/conf
Hadoop Jar and Library Files   /usr/lib/hadoop-0.20
Hadoop Log Files               /var/log/hadoop-0.20
Hadoop Man pages               /usr/share/man
Hadoop service scripts         /etc/init.d/hadoop-0.20-*
```

ANNEXE II

INSTALLATION ET CONFIGURATION DE HBASE

Installation de HBase

Étape 1 : Installation de HBase :

1. Installer HBase.
   ```
   $ sudo apt-get install hadoop-hbase
   ```
2. ILister les fichiers installés.
   ```
   $ rpm -ql hadoop-hbase
   ```

Étape 2 : Configurer la limite « ulimi » pour HBase :

1. Editer le fichier « /etc/security/limits.conf » avec la commande « sudo gedit »
   ```
   hdfs  -       nofile  32768
   hbase  -      nofile  32768
   ```
2. Pour appliquer les changements, éditer faites au fichier « /etc/security/limits.conf », ajouter la ligne suivante au fichier « /etc/pam.d/common-session file:
   ```
   session required  pam_limits.so
   ```

Étape 3 : Démarrer HBase en mode Autonome « Standalone »

1. Pour démarrer HBase en mode autonome, il faut en premier installer le Master pour HBase.
   ```
   $ sudo apt-get install hadoop-hbase-master
   ```
2. Démarrer le master
   ```
   $ sudo /etc/init.d/hadoop-hbase-master start
   ```
3. Verifier l'installation en navigaunt à l'adresse suivante : http://localhost:60010.

ANNEXE III

INSTALLATION ET CONFIGURATION DE POSTGRESQL SUR UBUNTU

Étape 1 : Installation de postgreSQL : [7]

1. Installer le serveur PostgreSQL.

```
$ sudo apt-get install postgresql postgresql-client postgresql-contrib
```

2. Installer pgadmin3. [8]

```
$ sudo apt-get install pgadmin3
```

Étape 2 : Configurer le compte du super usager : [9]

1. Réinitialiser le mot de passe du compte « admin ».

```
$ sudo su postgres -c psql template1
template1=# ALTER USER postgres WITH PASSWORD 'password';
template1=# \q
```

2. Change le mot de passe pour la base de données et pour l'usager « Postgres » de Linux.

```
$ sudo passwd -d postgres
$ sudo su postgres -c passwd
```

---

[7] http://engineering.gomiso.com/2011/07/28/adventures-in-scaling-part-2-postgresql/

[8] http://askubuntu.com/questions/79137/installing-pgadmin-3-for-ubuntu-11-10-and-postgresql-9-1

[9] http://www.molecularsciences.org/postgresql/installing_postgresql-8.4_on_ubuntu_10.x

ANNEXE IV

INSTALLATION ET CONFIGURATION DE SQOOP ET JDBC DRIVER POUR POSTGRESQL

Étape 1 : Installation de Sqoop : [10]

1. Installer Sqoop.
   ```
   $ sudo apt-get install sqoop
   ```

Étape 2 : Installation de Driver JDBC pour PostgreSQL :

1. Installer le driver JDBC3 pour PostgreSQL. [11]
   ```
   $ sudo apt-get install libpg-java
   ```
2. Copier le driver JDBC dans le répertoire «/usr/lib/sqoop/»

Étape 3 : Tester l'installation de Sqoop et du driver JDBC pour PostgreSQL:

1. Lister toutes les tables de la base de donnée de PostgreSQL avec la commande "list-tables" de Sqoop.

```
aklos@anna-klos:~$ sqoop list-tables --connect jdbc:postgresql://localhost:5432/20120224-BDS2D-New_Schema --password postgre
s --username postgres
12/03/25 21:16:17 WARN tool.BaseSqoopTool: Setting your password on the command-line is insecure. Consider using -P instead.
12/03/25 21:16:17 INFO manager.SqlManager: Using default fetchSize of 1000
chromosome
ngs_hg19_position
ngs_hg18_position
variant_detection_algorithm
ngs_position
screening_pipeline
sequence_aligner_algorithm
sequencer
variant_annotation_algorithm
ngs_variant
ngs_queryvariantposition
ngs_sample_variant
ngs_coverage
aklos@anna-klos:~$
```

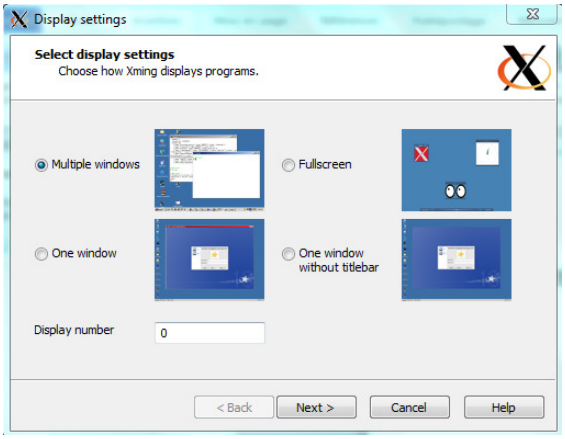
[10] https://ccp.cloudera.com/display/CDHDOC/Sqoop+Installation

[11] http://ubuntuguide.org/wiki/Ubuntu:Gutsy et https://issues.apache.org/jira/browse/SQOOP-390

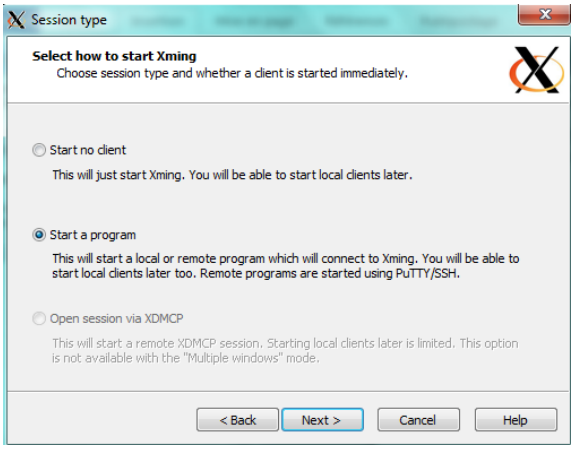
ANNEXE V

UTILISATION DE XMING

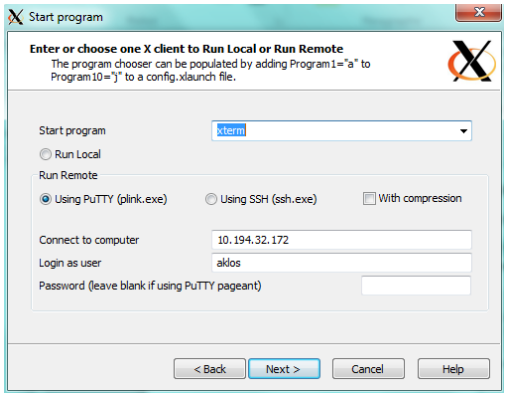
1. Démarrer XLaunch et sélectionner l'option « Multiple windows ». Ce mode vous permet d'ouvrir plusieurs fenêtres en même temps sur votre poste client.



2. Aller à l'étape suivante et sélectionner l'option « Start a program ». Ce mode vous permet d'ouvrir une session en passant par le protocole SSH.



3. Aller à l'étape suivante et sélectionner l'option « Run Remote – Using PuTTY». Cette option vous permet de spécifier l'adresse de l'ordinateur vers lequel vous voulez vous connecter.



4. Aller à l'étape suivante et patienter pendent que la connexion vers l'ordinateur est établie. Une fois c'est fait vous allez devoir saisir votre mot de passe pour ouvrir la console de votre ordinateur Ubuntu.