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

# SYS870
# PRÉPARATION D'UN LABORATOIRE BIG DATA À L'AIDE DE DOCKER

JULIEN BÉLIVEAU
BELJ22119008

DÉPARTEMENT DE GÉNIE LOGICIEL ET DES TI

**Professeur superviseur**

**ALAIN APRIL**

MONTRÉAL, 10 AOÛT 2015
ÉTÉ 2015

## REMERCIEMENTS

Je tiens à remercier mon professeur-superviseur Alain April. Il m'a donné la chance de travailler sur ce projet qui m'a appris beaucoup de choses. Il a aussi été compréhensif lorsque j'ai demandé d'avoir plus de temps pour terminer ce rapport.

David Lauzon est une autre personne envers laquelle je suis très reconnaissant. Il m'a guidé tout au long du projet et répondait à mes questions en cas de problème. Il a aussi su adapter les heures de nos rencontres en tenant compte de mon horaire de travail.

J'aimerais également à remercier Sébastien Bonami et François Langelier. Leur travail dans le cadre du projet Adamcloud m'a donné un exemple de comment bien travailler avec les fichiers Docker.

Finalement, je veux aussi mentionner Michaël Faille, étudiant au baccalauréat. Son arrivée à la fin du projet m'a permis de débloquer sur certains points techniques grâce à son expertise concernant Linux et Docker.

**SYS870**
**PRÉPARATION D'UN LABORATOIRE BIG DATA À L'AIDE DE DOCKER**

**JULIEN BÉLIVEAU**
**BELJ22119008**

## RÉSUMÉ

Ce rapport présente le travail qui a été fait dans le cadre d'un projet de fin d'études réalisé pour le cours « GTI792 – Projet de fin d'études en génie des TI » à l'École de technologie supérieure.

Le cours de maîtrise « SYS870 – Introduction au Big Data » va être enseigné pour la première fois à l'automne 2015. Il a été décidé qu'un laboratoire pratique serait pertinent pour tester la compréhension des étudiants. L'objectif du projet était d'aider les enseignants du cours à préparer l'infrastructure du laboratoire et son contenu.

Le laboratoire utilise la technologie de conteneur Docker pour l'hébergement de l'environnement où sont exécutées les applications. Les conteneurs existeront sur un serveur de l'ÉTS et une coordination avec l'école a été faite pour que les ressources soient prêtes et accessibles.

Une image Docker d'Hadoop a été peaufinée et une image de HBase a été créée. Les deux sont simples et offrent une persistance des données en cas de problème avec un conteneur. Les ports pour leur interface web sont accessibles. D'autres images ont été travaillées et pourraient être ajoutées dans le futur, mais elles ont été mises de côté par manque de temps.

Les données météorologiques GSOD ont été choisies comme *dataset* à avoir dans des tables HBase. Elles se séparent en une table data et une table station. Le téléchargement, la fusion et le formatage ont été automatisés dans un script. Les données ont également été converties au format CSV pour permettre l'importation dans HBase. Une tâche MapReduce a été choisie pour faire les importations.

Des questions variées à poser concernant les données ont été déterminées afin de forcer différentes conceptions de table. Des requêtes pour répondre à ces questions ont été créées et la conception de clé qu'elles utilisent a été présentée. La table data utilise la clé « stationID_date » ou « date_stationID » tandis que la table station utilise « stationID_wban » ou « country_state_stationID ». La réponse à la première question a été trouvée à l'aide de l'outil de ligne de commande HBase et les suivantes ont été trouvées avec l'aide de l'API Java. Des programmes Java ont donc été créés.

Le problème majeur qui a été rencontré est le fonctionnement d'un service YARN dans l'image Hadoop. L'importation des données dans HBase en dépendait et a beaucoup été délayée à cause du problème qui a finalement été réglé.

Finalement, pour les prochaines itérations du laboratoire, d'autres images comme Sqoop et Hive pourraient être ajoutés. Les données HBase qui résident localement sur le serveur pourraient aussi être mises dans HDFS. HBase pourrait être utilisé en mode distribué et un préfixe pourrait être ajouté à chaque clé pour empêcher un engorgement de trafic sur un serveur particulier.

**TABLE DES MATIÈRES**

Page

## LISTE DES TABLEAUX

Page

## LISTE DES FIGURES

Page

## LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

| | |
|---|---|
| API | Application Program Interface |
| CSV | Comma Separated Values |
| ÉTS | École de technologie supérieure |
| FTP | File Transfer Protocol |
| GSOD | Global Surface Summary of the Day |
| HDFS | Hadoop Distributed File System |
| NOAA | National Oceanic and Atmospheric Administration |
| PFE | Projet de fin d'études |
| TI | Technologies de l'Information |
| VM | Virtual Machine |
| YARN | Yet Another Resource Manager |

## LISTE DES SYMBOLES ET UNITÉS DE MESURE

| | |
|---|---|
| Mo | Mégaoctet |

# INTRODUCTION

Le concept de Big Data, aussi connu sous le nom de mégadonnées, est de plus en plus important dans le monde de l'informatique. Il représente un ensemble de données trop volumineuses pour être gérées par des outils de bases de données et d'entreposage traditionnels. Beaucoup d'industries se retrouvent maintenant avec ce problème et ont besoin de nouvelles technologies pour gérér leurs données.

Consciente de ce domaine en croissance, l'École de technologie supérieure a adapté son offre de cours des TI et a notamment créé le cours SYS870 à la maîtrise qui fait l'introduction au monde du Big Data. L'arrivée du cours a amené le besoin d'avoir un travail de laboratoire pour les étudiants. Ce projet porte sur la préparation du laboratoire utilisant entre autre la technologie de conteneurs Docker et une base de données non relationnelle.

Ce document permettra d'expliquer ce qui a été fait en détails dans le projet. Il comporte plusieurs chapitres. Tout d'abord, une mise en contexte explorera les besoins du cours et les concepts présents dans le PFE. Par la suite, les images Docker seront présentées. La section suivante portera sur les données utilisées pour la base de données non relationnelle. Elle sera suivie de l'explication de ce qui sera demandé au laboratoire. Ensuite, les problèmes rencontrés lors du projet seront présentés et ils seront finalement suivis d'une liste des travaux futurs pouvant être réalisés.

# CHAPITRE 1

# MISE EN CONTEXTE

## 1.1 Big Data

Le terme Big Data ou mégadonnées définit un rassemblement de données qui est si grand qu'il peut difficilement être géré en utilisant des techniques traditionnelles [1]. Toutes les données qu'ont les grands réseaux sociaux en sont un bon exemple. Les alternatives créées avec l'arrivée de Big Data concernent le traitement des données, leur entreprosage, leur analyse et leur création.

Parmis les technologies de Big Data, il y a Hadoop. Développé par la fondation Apache, c'est un *framework* pour l'entreposage et le traitement de larges données. Il utilise un *filesystem* nommé HDFS qui permet de distribuer la charge des données sur plusieurs serveurs et une mise à l'échelle facile.

Une autre partie importante de Big Data est composée des bases de données non relationnelles qui sont aussi connues sous le nom NoSQL. La structure de données qu'elles utilisent leur permet de faire de la mise à l'échelle à l'horizontal. Cela signifie que pour augmenter les performances, il suffit d'ajouter des serveurs standards à un *cluster*. Cela diffère des BD traditionnelles où les serveurs doivent généralement être remplacés par des machines plus puissantes, des superordinateurs, pour améliorer les performances. MongoDB et HBase sont deux des produits NoSQL les plus populaires.

## 1.2 Adamcloud

Adamcloud est un projet faisant l'analyse du génome humain afin de diagnostiquer des maladies chez des patients [2]. Il utilise des outils développés par l'université Berkeley en Californie et des outils Big Data comme Hadoop. L'ÉTS a été un grand collaborateur dans la dernière année.

Le projet utilise la technologie de conteneurs Docker et des images ont donc dû être créées pour les différentes applications. L'utilisation de Docker a permis de démontrer le potentiel de cette technologie et a augmenté l'appréciation du produit pour les gens impliqués dans le projet.

## 1.3 Cours SYS870

Compte tenu de la présence de plus en plus importante des mégadonnées dans le monde des TI, le cours de maîtrise SYS870 a été créé. Intitulé « Introduction au Big Data », il vise à présenter aux étudiants les concepts de base et les technologies actuelles de l'écosystème. Le cours sera donné pour la première fois à l'automne 2015 et les enseignants David Lauzon, Alain April et Olivier Mirandette se répartiront les séances.

Les objectifs du cours sont les suivants [3] :

- Comprendre les limites des bases de données relationnelles pour traiter de grandes quantités de données;
- Comprendre les forces et faiblesses des systèmes NoSQL / Big Data, et de savoir quand les utiliser et ne pas les utiliser;
- Être capable de porter un jugement sur une nouvelle technologie NoSQL selon les critères définis en classe;
- Déterminer quelle(s) technologie(s) utiliser selon une mise en contexte, et de justifier les choix technologiques.

## 1.4 Problématique

Afin de répondre aux objectifs du cours, il est important que les étudiants puissent mettre en pratique les notions qu'ils apprennent. Il est donc devenu évident qu'un laboratoire devait être mis en place.

L'utilisation d'une machine virtuelle par équipe avait été testée dans un autre cours et n'avait pas été un succès. La VM était d'une grande taille et donc longue à transférer sur les ordinateurs personnels des équipes. Ses performances dépendaient aussi du poste utilisé.

Étant donné la bonne expérience vécue avec Docker dans le cadre d'Adamcloud [4], le produit a été choisi comme alternatives à la virtualisation. De plus, il a été décidé qu'un serveur de l'école serait mis à la disposition des étudiants pour garantir un environnement uniforme pour chaque équipe, de la performance et de la stabilité.

## 1.5 Objectifs

Alors qu'il y avait eu initialement des discussions à propos la continuation d'Adamcloud, l'objectif du projet est devenu la préparation de l'infrastructure pour le laboratoire de SYS870. David Lauzon était le client principal qui faisait part de ses besoins. Il fallait mettre en place des images Docker d'applications Big Data et définir ce qui serait demandé comme travail aux étudiants dans les conteneurs.

Il fallait également assurer la coordination avec l'ÉTS pour la préparation d'un serveur accessible par les étudiants de manière à leur offrir un espace de travail fiable disponible dès le premier jour.

# CHAPITRE 2

# IMAGES DOCKER

## 2.1 Hadoop

Une des images importantes à livrer était celle de Hadoop. Elle avait déjà été créée dans le cadre d'Adamcloud [5], mais il restait des points à corriger et tester.

Le point le plus important était la persistance des données. L'instruction « VOLUME » pour le répertoire des données était présente dans le *dockerfile*, mais ce n'était pas suffisant. Elle crée un *mount* sur l'hôte, mais dans un dossier qu'on ne peut spécifier et nommé selon le conteneur actuel. La suppression d'un conteneur et la création d'un nouveau crée donc un nouveau répertoire pour le *mount*.

| Rôle | Conteneur | Dossier (conteneur) | Dossier (hôte) |
|---|---|---|---|
| HDFS namenode | cde167197[…] | /data | /var/lib/docker/vfs/dir/cde167197[…] |
| HDFS namenode (nouvelle instance remplaçant l'ancienne) | liw629431[…] | /data | /var/lib/docker/vfs/dir/adc629431[…] |

**Tableau 1 Volumes en utilisant la commande VOLUME**

L'ajout du paramètre « -v » à la commande « docker run » lors de la création des conteneurs règle le problème. Elle permet de spécifier la destination du *mount* sur l'hôte. De cette façon, nous pouvons nous assurer que les données à conserver des instances d'un conteneur seront toujours au même endroit. La documentation sur le GitHub a donc été mise à jour pour refléter ce changement. Par exemple, elle indique que la création d'un conteneur *datanode* se fait avec la commande suivante :

*docker run -d --name hdfs-datanode1 -h hdfs-datanode1 -p 50075:50075* ***-v /hostdirectory/docker-volumes/hdfs-datanode1:/data*** *--link=hdfs-namenode:hdfs-namenode --link=hdfs-secondarynamenode:hdfs-secondarynamenode gelog/hadoop:2.6.0 hdfs datanode*

| **Rôle** | **Conteneur** | **Dossier (conteneur)** | **Dossier (hôte)** |
|---|---|---|---|
| HDFS datanode 1 | cde167197[…] | /data | /hostdirectory/docker-volumes/hdfs- datanode1 |
| HDFS datanode 1 (nouvelle instance remplaçant l'ancienne) | liw629431[…] | /data | /hostdirectory/docker-volumes/hdfs- datanode1 |

**Tableau 2 Volumes en utilisant le paramètre –v**

Un autre élément qui a dû être changé pour que le *mount* soit fonctionnel est le répertoire des fichiers de configuration. Il était initialement dans « /data », mais il n'est pas possible d'utiliser l'instruction « ADD » dans un *dockerfile* pour ajouter un fichier à un volume. L'emplacement a donc été changé pour être celui par défaut, soit « /usr/local/hadoop/etc/hadoop ».

En ce qui concerne l'utilisation de l'interface web pour la gestion de HDFS, seul le port exposé pour le *datanode* était incorrect. Il a donc été changé de 50081 à 50075.

## 2.2 HBase

La seconde image cruciale était celle de HBase. Elle a été créée à partir de la version 1.1.1 et hérite de l'image Ubuntu-Java. Elle fonctionne en mode *standalone*. Il y a donc un serveur pour tout gérer et les données sont entreposées sur le *filesystem* local. Cela a été choisi pour la simplicité de la solution.

En ce qui concerne la persistance des données, la même logique que pour l'image Hadoop a été utilisée. Un *mount* du dossier « /data » qui contient les données HBase est créé sur l'hôte. De cette façon les tables ne sont plus à la merci du conteneur.

Les ports de HBase sont exposés dont le 16010 qui permet d'accéder à l'interface web. La commande permettant de lancer un conteneur HBase est donc la suivante :

> *docker run -d --name hbase-master -h hbase-master -p 16010:16010 -v /hostdirectory/docker-volumes/hbase-master:/data gelog/hbase*

## 2.3 Autres

Au cours de la session, il y a eu un début de travail fait sur d'autres images Docker qui auraient pu être pertinentes pour le cours. La première était pour CDH qui est une distribution d'Apache Hadoop faite par Cloudera. La configuration du *repository* de Cloudera était simple à faire, mais l'installation des composantes était problématique. Par exemple, mettre en place manuellement Impala dans CDH nécessitait Hive qui demandait Impala qui nécessitait MySQL ou PostgreSQL [6] . Étant donné le temps limité, il a été décidé de mettre cette image de côté pour concentrer les efforts sur Hadoop et HBase.

Sqoop a également été mentionné en fin de projet. C'est une application permettant de transférer des données d'une base de données relationnelle dans HDFS ou directement dans une table HBase [7]. Des tests avec une ébauche de *dockerfile* ont été faits, mais il y avait

une erreur au démarrage du service. Puisqu'il restait peu de temps au projet et que d'autres enjeux techniques étaient encore présents, il n'y a pas eu de *troubleshooting* avancé qui a été fait.

Finalement, des ébauches de *dockerfile* pour Redis et MongoDB ont aussi été réalisées, mais elles ont été abandonnées lorsqu'il a été confirmé qu'il y avait déjà une image fonctionnelles pour ces deux application.

# CHAPITRE 3

# DONNÉES GSOD

## 3.1 Contexte

Les données GSOD (Global Surface Summary of the Day) sont des mesures météorologiques provenant de la NOAA (National Oceanic and Atmospheric Administration). Des milliers de stations enregistrent chaque jour des informations concernant la température, la quantité de pluie, la visibilité, etc [8]. Des données sont recueillies depuis le début du 20$^{e}$ siècle.

Étant donné le grand nombre d'enregistrements sur une longue période de temps, le cas des données GSOD a été choisi comme étant un bon candidat pour être aidé par des technologies Big Data. Le but était donc d'avoir ultimement les données dans une BD HBase.

Les données sont rassemblées essentiellement sous l'équivalent de deux tables (voir l'annexe V). La première contient les données météorologiques de chaque station et sera surnommée data. La seconde contient des détails géographiques concernant chaque station et sera appelée station.

## 3.2 Téléchargement

Le premier défi était de récupérer les données. Elles sont partagées par la NOAA via un site FTP public. Un dossier est présent pour chaque année et chaque dossier contient un fichier compressé par station. Ce fichier contient toutes les mesures d'une station donnée pour l'année complète. Il y a au plus 365 entrées, soit une par jour. Un script *bash*, adapté d'un projet existant [9], télécharge les données avec la commande *wget* et les décompresse pour une année précise. Un autre script appelle le premier pour chaque année d'un intervalle donné en paramètre (voir l'annexe VI). Les enregistrements de toutes les stations pour une année représentent un fichier d'environ 500 Mo.

### 3.3 Script maître

Afin de simplifier l'exécution de tous les scripts, un script maître a été créé. Il s'occupe de faire l'appel de tous les autres dans le bon ordre. Intitulé « gsod_master_script.sh », il faut lui donner en paramètre l'année de début et l'année de fin à récupérer. Par exemple, pour avoir les données de 1990 à 2014, il suffit de lancer la commande « ./gsod_master_script.sh 1990 2014 ». Le résultat est un fichier rassemblant toutes les données dans un seul. Un fichier par année est également créé pour permettre de faire des tests moins lourds sur des données précises.

### 3.4 Formatage

Une fois les données téléchargées et rassemblées dans un fichier, il fallait les formater. Les différentes colonnes étaient séparées par des espaces qui n'étaient pas de la même grandeur pour toutes les variables. Il a été décidé de les remplacer par des virgules pour avoir un fichier au format CSV et donc un séparateur commun.

L'outil in2csv sur Linux a été utilisé. Il permet de faire la conversion en spécifiant le fichier source et un fichier de schéma. Ce dernier définit les différentes colonnes et l'emplacement de chacune. Par exemple, la température moyenne commence à la position 24 et a une longueur de six caractères pour chaque ligne. La conversion peut durer plusieurs minutes pour une seule année.

Une fois que le fichier était converti en CSV, il fallait le formater selon la *rowkey* choisie. Cela se traduisait par exemple par la transformation d'une virgule en un sous-tiret à l'aide du programme *sed* afin de fusionner deux colonnes. La conception des clés sera vue en détail dans un prochain chapitre.

### 3.5 Importation

Afin d'importer les fichiers CSV dans HBase, la tâche Map Reduce importTSV a été utilisée. Il faut spécifier le fichier source, la table de destination, le séparateur, les noms des colonnes et la clé [10]. La commande peut par exemple prendre la forme suivante pour les deux tables GSOD :

```
hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -
Dimporttsv.columns="HBASE_ROW_KEY,d:name,d:ctry,d:st,d:lat,d:lon,d
:elev,d:begin,d:end" -Dimporttsv.separator="," gsod_stations
file:///data/persistent/hbase/import/gsod_stations.csv
```

```
hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -
Dimporttsv.columns="HBASE_ROW_KEY,d:wban,d:temp,d:count_temp,d
:dewp,d:count_dewp,d:slp,d:count_slp,d:stp,d:count_stp,d:visib,d:count_v
isib,d:wdsp,d:count_wdsp,d:mxspd,d:gust,d:max,d:flag_max,d:min,d:flag_
min,d:prcp,d:flag_prcp,d:sndp,d:fog,d:rain_drizzle,d:snow_ice_pellets,d:h
ail,d:thunder,d:tornado_funnel_cloud" -Dimporttsv.separator="," 
gsod_data_stn_date
file:///data/hbase/gsod_import/gsod_data_2000_to_2014_stn_date.csv
```

L'importation dure une dizaine de minutes pour une seule année. Dans le cas où plusieurs conceptions seraient choisies pour les données météorologiques, il faudrait répéter l'importation dans une nouvelle table pour chaque variante.

### 3.6 Étendue

En faisant les tests de conversion et d'importation des données, il est devenu évident que l'utilisation de l'ensemble des données GSOD serait problématique. Pour une seule année, la conversion en CSV suivie de l'importation peut durer près de vingt minutes. Nous savons également que la plupart des étudiants devront faire quelques essais erreurs avant d'obtenir un bon résultat. De plus, ils auront probablement plusieurs variantes de la table data. Il a

donc été décidé d’utiliser les années 2000 à 2014 et il sera recommandé aux équipes de faire leurs tests initiaux avec une seule année.

# CHAPITRE 4

# LABORATOIRE

## 4.1 Questions

Les questions candidates pour le laboratoire ont été déterminées avec l'aide de David Lauzon. Elles portent sur les données GSOD et doivent être répondues en faisant des requêtes sur une base de données HBase. L'objectif était de forcer les étudiants à faire des requêtes variées nécessitant des conceptions de table différentes selon les questions. Elles sont les suivantes :

1) Quelles sont toutes les données météo de la station SORKJOSEN pour le 25 décembre 2014?
2) Quelle a été la température minimale, moyenne et maximale au Rhode Island entre le 15 mai et le 14 avril 2002?
3) Nommez les stations où il a fait en moyenne moins de 5 F le 2 février 2008
4) Dans la région de Montréal ([45.405] à [45.705], [-73.47] à [-73.98]), nommez les occurrences (station et date) où il y a eu de la grêle entre 2000 et 2014.
5) Quelles sont les températures en Celsius de la station 008414 en 2014? Bonus : Convertissez les données dans la table.

Les questions sont ordonnées de manière à avoir une difficulté croissante et afin de proposer un défi aux étudiants de toutes les forces.

## 4.2 Conception des tables

Dans le modèle utilisé par HBase, il y a le concept de familles de colonnes [11]. Ces dernières sont des regroupements de colonnes qui sont généralement de la même nature. Étant donné que toutes les données sont de type météorologiques pour la table data et principalement géographiques pour la table station, une seule famille a été utilisée pour chaque table. Apache recommande d'ailleurs d'utiliser au maximum trois familles et une

seule si possible. La famille de chaque table se nomme « d » afin d'occuper le moins d'espace possible.

Afin de répondre aux questions, deux conceptions ont été choisies par table. Dans le cas de la table station, c'est une clé formée des colonnes pays, état et *stationID* qui a été choisie. Son format peut être par exemple : « US_RI_720033 ». Cette clé est optimisée pour toutes les requêtes en lien avec le pays ou l'état. Pour récupérer toutes les stations du Rhode Island, il suffit par exemple de faire un « scan » sur la plage « US_RI » à « US_RJ ». Un balayage pour le Canada serait fait avec « CA » comme clé de début et « CB » comme clé de fin. Cette conception est utilisée pour les questions 2 et 4 étant donné qu'elles font références à un emplacement dans leurs critères.

La seconde version de la table station utilise une clé combinant *stationID* et le numéro WBAN. Elle peut par exemple être « 480172_99999 ». Elle est faite pour les requêtes sur une station précise lorsque son identifiant est connu. Ce modèle est utilisé pour les questions 1 et 4.

Pour la table data, la première conception de clé rassemble le *stationID* et la date de prise de mesure. Elle peut par exemple être : « 480172_20140522 ». Cela a été pensé en fonction de requêtes de mesures pour une station précise pour une période de temps donnée. Il est donc facile d'obtenir tous les enregistrements de la station 480172 en faisant simplement un « scan » de la clé « 480172* » à « 480173* ». Pour obtenir toutes les données de la station entre 2006 et 2008, la plage serait « 480172_2006* » à « 480172_2009* ». La même logique peut être utilisée pour une plage de mois ou de journées d'un mois. Cette conception est utilisée pour les questions 1, 2, 4 et 5.

L'autre variante de la table data utilise une clé qui rassemble la date puis le *stationID*. Elle peut par exemple être : « 20140522_480172 ». Cela permet de faire des requêtes concernant les données de toutes les stations pour une période de temps donnée. Un *scan* de la colonne « 20140507* » à « 20140514* » nous donne facilement toutes les mesures de toutes les

stations pour la semaine spécifiée. La période peut être une année, un mois ou des journées. Cette conception est utilisée pour la question 3.

### 4.3 Critères de correction

Plusieurs critères seront considérés pour l'évaluation du laboratoire. Le premier sera concernant l'importation des données. Est-ce que toutes les données GSOD demandées sont dans la base de données HBase?

Le deuxième sera concernant les tables qui auront été créées et leurs *rowkeys*. Est-ce qu'il y a une seule version des tables data et stations pour toutes les questions? Quelle *rowkey* a été choisie pour chaque table à chaque question?

L'étendue des requêtes sera aussi analysée. Idéalement, il ne faut pas faire un *scan* sur l'ensemble des rangées. Il est possible de déterminer une clé de début et une de fin.

Ces éléments auront tous un impact sur la performance des requêtes. Le temps de réponse sera pris en compte. La réponse aux requêtes devrait être instantanée à l'œil humain. Le contraire démontrerait une mauvaise optimisation.

Finalement, l'exactitude des réponses sera vérifiée. Une grille de correction sera faite avant le début du cours afin d'assigner une pondération à chaque critère.

### 4.4 HBase Shell

Le « HBase Shell » est l'outil de gestion de la base de données en ligne de commande. Comparé aux requêtes SQL possibles, il est extrêmement limité [12]. Les commandes utilisées pour le laboratoire sont peu variées. Les tables ont été créées à l'aide de la commande « create ». L'inventaire des tables était vérifié avec la commande « list ». Les

deux instructions les plus importantes sont « get » et « scan ». La première permet de sortir toutes les valeurs d'une rangée pour une clé donnée. La seconde retourne les valeurs de toutes les rangées d'une plage spécifiée. Ce sont principalement des *scans* qui ont été utilisés dans les requêtes étant donné que les questions portent sur plusieurs entrées.

Afin de répondre à la première question, cette première requête été utilisée :

```
scan 'gsod_stations', {FILTER =>
SingleColumnValueFilter.new(Bytes.toBytes('d'), Bytes.toBytes('name'),
CompareFilter::CompareOp.valueOf('EQUAL'),
BinaryComparator.new(Bytes.toBytes('SORKJOSEN')))}
```

Elle a été choisie pour démontrer que des filtres peuvent être utilisés [13] lorsque la conception de la clé ne correspond pas exactement à ce qui est demandé. Le *scan* passe à travers toutes les rangées de la table, mais seule la rangée avec le nom SORKJOSEN est affichée grâce au filtre. Le résultat permet d'avoir le numéro de station et donc de faire la requête suivante qui répond à la question :

```
get  'gsod_data_stn_date', '010460_20141225'
```

### 4.5 Programmes Java

Pour répondre aux questions 2 à 5 qui sont plus complexes, l'API Java a été choisie étant donné qu'elle permet de passer facilement à travers des boucles, de gérer les conditions et d'afficher les informations de façon plus personnalisée [14]. Un programme par question a donc été conçu puisqu'elles sont toutes indépendantes les unes des autres. Ils se nomment « HbaseQueryQ2.java » à « HbaseQueryQ5.java » et sont consultables à l'annexe IX.

La logique est semblable pour les quatre programmes. Des éléments du paquet « org.apache.hadoop.hbase » sont tout d'abord importés. L'instanciation des tables data et station est ensuite faite. Selon la question, une première table est alors parcourue à l'aide

d'une boucle d'un *scan* avec les critères demandés et les résultats sont utilisés pour parcourir la seconde table, vérifier les critères spécifiés et afficher la réponse finale. Un « System.out.println » a été choisi pour son aspect simple et efficace.

Afin d'exécuter un programme Java dans le conteneur HBase, il faut d'abord le compiler :

*javac -cp .:$(hbase classpath) HbaseQueryQ2.java*

Par la suite, la commande suivante est utilisée lancer l'application :

*java -cp .:$(hbase classpath) HbaseQueryQ2*

Les résultats des requêtes sont alors affichés en texte dans le terminal.

## 4.6 Coordination avec l'ÉTS

Tout au long du projet, il y a eu plusieurs échanges avec Patrice Dion qui est chargé de l'application technologique et informatique à l'ÉTS. Nous nous sommes assurés qu'un serveur de l'école avec Docker installé et des ressources adéquates serait accessible par les étudiants avec des droits suffisants.

# CHAPITRE 5

# PROBLÈMES RENCONTRÉS

## 5.1 Fonctionnement de YARN sur l'image Hadoop

Le plus grand problème lors du projet a été le fonctionnement de YARN à partir de l'image Hadoop. Il a été très dérangeant étant donné que l'importation des données GSOD dans HBase se faisait à partir d'une tâche « MapReduce » et utilisait donc YARN. L'avancement du projet a donc été sérieusement impacté pendant la durée du problème.

Le premier blocage était que le service « Node Manager » ne démarrait pas. C'est un élément crucial de YARN qui vérifie l'utilisation des ressources sur chaque *node* et qui se rapporte au « Resource Manager » [15]. Tous les fichiers semblaient être bons ce qui rendait le *troubleshooting* plus difficile. Après avoir refait un *dockerfile* Hadoop au complet et à partir d'une autre référence, la cause s'est révélée. Le fichier de configuration « capacity-scheduler.xml » qui est nécessaire était présent, mais il n'était pas pris en compte étant situé au mauvais endroit. Lorsque le répertoire des fichiers de configuration a été changé de « /data/conf » à « /usr/local/hadoop/etc/hadoop » tel que mentionné en 2.1, le problème a été réglé.

Le service « Node Manager » démarré permettait de lancer des tâches d'importation, mais elles demeuraient toutes infiniment à 0%. L'utilisation des valeurs par défaut du fichier de configuration « mapred-site.xml » a permis de régler le problème.

Ces problèmes ont causé un changement de l'image HBase. Alors qu'elle héritait au départ de l'image Hadoop, il a été décidé de la faire hériter directement de Java pour sauter le problème même si ce dernier a ultimement été réglé.

## 5.2 Ports HBase

Initialement, les tests d'accès à l'interface web de HBase n'étaient pas fonctionnels même si les bons ports semblaient être exposés. Après quelques recherches, il a été découvert que les ports utilisés par HBase ont changé après la version 0.98 [16], mais la plupart de la documentation fait encore référence aux anciens. L'image HBase a été changée pour refléter les changements. Voici les nouveaux ports :

hbase.master.port : 60000 -> 16000
hbase.master.info.port (http): 60010 -> 16010
hbase.regionserver.port : 60020 -> 16020
hbase.regionserver.info.port (http): 60030 -> 16030
hbase.status.multicast.port : 60100 -> 16100

## 5.3 Problèmes organisationnels

La gestion du temps et des problèmes auraient pu être meilleures pendant le projet. Par exemple, le problème avec YARN a beaucoup ralenti le projet et aurait pu être priorisé davantage. De plus, la sélection des données GSOD pour la base de données HBase s'est faite relativement tard dans le projet. La conception des méthodes de récupération des données, d'importation et des clés de tables s'est donc réalisée principalement dans la deuxième moitié de la session.

Il y a également du temps qui a été mis en début de projet sur Adamcloud avec beaucoup de lecture et des tests sur un « cluster » de Mac minis. Le temps n'a pas été complètement perdu étant donné que les notions Big Data étaient présentes aussi dans ce projet, mais il aurait pu être mieux distribué puisqu'Adamcloud a été mis de côté.

Finalement, la préparation avec Patrice Dion aurait pu être faite plus tôt pour éviter entre autre les conflits d'horaire et les retards causés par les vacances.

# CHAPITRE 6

# TRAVAUX FUTURS

## 6.1 Intégration de HBase à HDFS

L'image actuelle de HBase entrepose sa base de données localement. Un aspect pertinent serait de l'intégrer à HDFS et donc aux conteneurs Hadoop. Ce scénario serait plus compliqué à mettre en place, mais il serait plus représentatif de ce qui est fait dans les vraies implémentations.

## 6.2 Utilisation de HBase en mode distribué

La configuration a été faite avec un seul serveur HBase en place qui a le rôle « master ». Il serait intéressant d'avoir plusieurs serveurs se répartissant la charge.

Dans ce scénario, il y aurait un serveur « master » qui surveillerait quelques serveurs « region ». Ces derniers seraient des « slaves » servant à entreposer chacun une partie des données [17].

## 6.3 Ajout d'un « salt » aux clés HBase

En ce moment, toutes les *rowkeys* pensées contiennent explicitement les données. Exemple : « 20141225_010460 ». Les données des tables HBase sont entreposées en ordre alphabétique de clé. Une région contient donc des entrées ayant des clés similaires en ce qui concerne l'alphabet. Dans un scénario où il y aurait plusieurs serveurs « region », cela pourrait causer un problème nommé « region server hotspotting ». Par exemple, si beaucoup de requêtes étaient faites pour une date spécifique, elles seraient toutes envoyées au même serveur

puisque les entrées « 20141225_010460 », « 20141225_010461 » et « 20141225_010462 » se suivent.

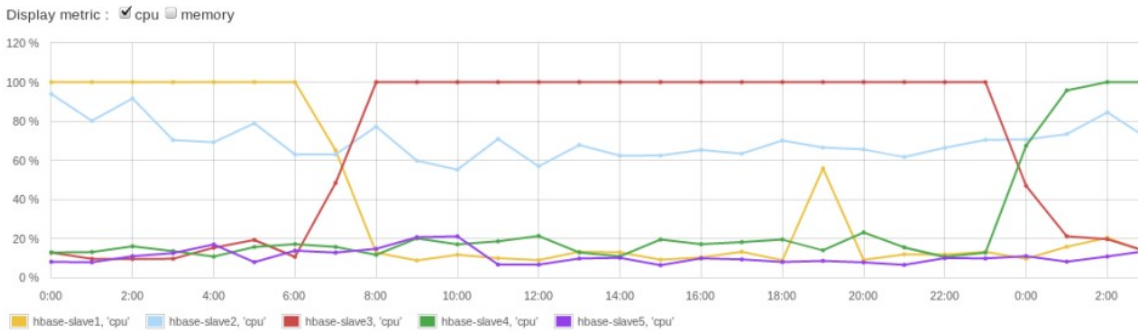


**Figure 1 Charges de serveurs HBase mal balancés [18]**

Afin de s'assurer que les requêtes soient envoyées de façon égale aux différents serveurs, il faudrait ajouter un *salt* devant chaque clé. Cela consiste en un préfixe qui permettrait de séparer les entrées des clés consécutives. Il serait important qu'il ne soit pas aléatoire et qu'il soit décodable pour conserver la possibilité de faire un *scan* sur une plage précise.

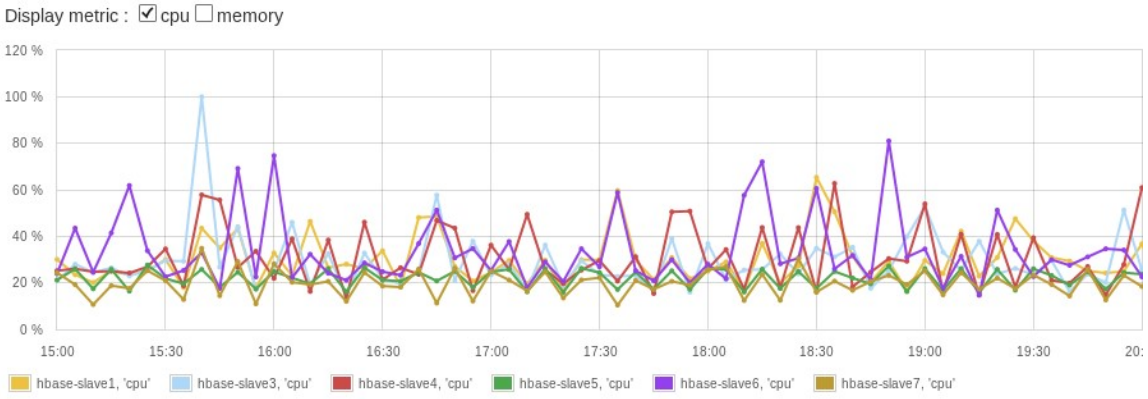


**Figure 2 Charges de serveurs HBase bien balancés [18]**

## 6.4 Ajout de nouvelles images Docker

Dans les futures itérations du cours, il pourrait y avoir davantage d'images Docker avec lesquelles les étudiants pourraient travailler. Nous pouvons penser notamment à Sqoop qui

permet de faire le transfert de données entre une base de données relationnelle et HDFS. Les étudiants pourraient par exemple avoir à importer les données GSOD dans une BD postgreSQL et ensuite utiliser Sqoop pour copier le tout dans HBase. Cela leur permettrait de comparer les performances et la structure de requête de deux types de base de données pour un même *dataset*.

Une autre image qui avait été discutée est celle de Hive. C'est une plateforme offrant un langage similaire à SQL nommé HQL pour interroger des données dans Hadoop [19]. Elle a été mise de côté à cause d'un simple manque de temps, mais elle devrait être facile à mettre en place.

# CONCLUSION

Ce rapport a présenté le travail qui a été fait dans le cadre d'un projet de fin d'études. Le but était de préparer un laboratoire pour le cours SYS870 à l'automne 2015. Deux images Docker fonctionnelles ont été livrées : Hadoop et HBase. Les deux sont simples et offre une persistance des données en cas de perte d'un container. En ce qui concerne la base de données HBase, des scripts de téléchargement et de formatage des données météorologiques GSOD ont été créés et une manière d'importer les données a été présentée. Des questions à demander sur les données ont été déterminées et des critères pour la correction ont été définis. Des programmes Java ont été développés qui fournissent les réponses aux questions et des références pour l'évaluation des laboratoires des étudiants.

Au plan personnel, ce fut un grand défi. La courbe d'apprentissage concernant les technologies Big Data, les bases de données non relationnelles et Docker est assez importante. Il y a donc eu une longue période de prise de connaissances et d'adaptation nécessaire avant de pouvoir beaucoup progresser.

Ce n'est que la première itération du cours et du laboratoire. Ils évolueront certainement lors des prochaines sessions et il ne serait pas surprenant que d'autres technologies soient ajoutées au programme.

## RECOMMANDATIONS

Au courant de la session, il y a eu un moment où le site FTP des données GSOD n'a pas été disponible pendant une journée entière. L'incident n'a pas bloqué le projet étant donné qu'une partie des données avait déjà été récupérée, mais cela a mis en évidence le fait que le laboratoire est dépendant de données hébergées à l'externe sur des ressources que nous ne contrôlons pas. Ce pourrait donc être une bonne idée d'avoir toujours de côté une partie du *dataset* sur un espace que nous gérons. Le site *Google Drive* du projet serait un bon espace.

## LISTE DE RÉFÉRENCES


[1] **Wikipedia**, Big Data
Internet, consulté le 11 mai 2015.
https://en.wikipedia.org/wiki/Big_data

[2] **LANGELIER François**, Adamcloud : Partie II
Rapport de projet, consulté le 4 mai 2015.

[3] **LAUZON David**, SYS870-A15
Syllabus, consulté le 13 mai 2015.

[4] **BONAMI Sébastien**, Adamcloud
Rapport de projet, consulté le 21 avril 2015.

[5] **GitHub**, GELOG/docker-ubuntu-hadoop
Internet, consulté le 26 mai 2015.
https://github.com/GELOG/docker-ubuntu-hadoop

[6] **Cloudera**, Installing and Deploying CDH Using the Command Line
Internet, consulté le 24 mai 2015.
http://www.cloudera.com/content/cloudera/en/documentation/core/latest/topics/cdh_ig_command_line.html

[7] **Wikipedia**, Sqoop
Internet, consulté le 15 juillet 2015.
https://en.wikipedia.org/wiki/Sqoop

[8] **NOAA**, GSOD FTP
Internet, consulté le 28 juin 2015.
ftp://ftp.ncdc.noaa.gov/pub/data/gsod/

[9] **GitHub**, tothebeat/noaa-gsod-data-munging
Internet, consulté le 3 juillet 2015.
https://github.com/tothebeat/noaa-gsod-data-munging/

[10] **Microsoft MSDN**, Loading data in HBase Tables using built-in ImportTsv utility
Internet, consulté le 28 juin 2015.
http://blogs.msdn.com/b/bigdatasupport/archive/2014/12/12/loading-data-in-hbase-tables-on-hdinsight-using-bult-in-importtsv-utility.aspx

[11] **Apache**, Apache HBase Reference Guide
Internet, consulté le 15 mai 2015.
http://hbase.apache.org/book.html#number.of.cfs

[12] **MAPR**, Getting Started with HBase Shell
Internet, consulté le 15 mai 2015.
https://www.mapr.com/products/mapr-sandbox-hadoop/tutorials/tutorial-getting-started-with-hbase-shell

[13] **Learn HBase**, HBase shell commands
Internet, consulté le 15 mai 2015.
https://learnhbase.wordpress.com/2013/03/02/hbase-shell-commands/

[14] **ALTAMIRA**, Handling Big Data with HBase Part 4: The Java API
Internet, consulté le 25 juillet 2015.
https://www.altamiracorp.com/blog/employee-posts/handling-big-data-with-hbase-part-4-the-java-api

[15] **NIXUS**, An Insight into Hadoop YARN NodeManager
Internet, consulté le 28 juin 2015.
http://nixustechnologies.com/2014/05/an-insight-into-hadoop-yarn-nodemanager/

[16] **Apache**, HBASE-10123
Internet, consulté le 26 juillet 2015.
https://issues.apache.org/jira/browse/HBASE-10123

[17] **Pivotal Support**, Understand Master Servers, Region Servers and Regions
Internet, consulté le 12 juillet 2015.
https://support.pivotal.io/hc/en-us/articles/200950308-HBase-Basics

[18] **Sematext**, HBaseWD: Avoid RegionServer Hotspotting Despite Sequential Keys
Internet, consulté le 22 juillet 2015.
http://blog.sematext.com/2012/04/09/hbasewd-avoid-regionserver-hotspotting-despite-writing-records-with-sequential-keys/

[19] **Apache**, APACHE HIVE
Internet, consulté le 29 juillet 2015.
https://hive.apache.org/

## BIBLIOGRAPHIE


**Docker**, Build, Ship and Run Any App, Anywhere
Internet, consulté le 4 mai 2015.
https://www.docker.com/

**Wikipedia**, Docker (software)
Internet, consulté le 21 avril 2015.
http://en.wikipedia.org/wiki/Docker_%28software%29

**Ubuntu**, The leading OS for PC, tablet, phone and cloud, Ubuntu
Internet, consulté le 5 mai 2015.
http://www.ubuntu.com/

**NOAA**, Global Summary of the Day (GSOD)
Internet, consulté le 28 juin 2015.
http://www7.ncdc.noaa.gov/CDO/cdoselect.cmd?datasetabbv=GSOD&countryabbv=&georegionabbv=

**Stackoverflow**, How can I load weather data into BigQuery
Internet, consulté le 30 juin 2015.
https://stackoverflow.com/questions/28161707/how-can-i-load-weather-data-into-bigquery

**YAHOO! DEVELOPER NETWORK**, Module 2: The Hadoop Distributed File System
Internet, consulté le 28 juin 2015.
https://developer.yahoo.com/hadoop/tutorial/module2.html

**Stackoverflow**, docker mounting volumes on host
Internet, consulté le 10 juillet 2015.
https://stackoverflow.com/questions/25311613/docker-mounting-volumes-on-host

**Highly Scalable Systems**, Hadoop Installation Tutorial (Hadoop 2.x)
Internet, consulté le 2 juillet 2015.
http://www.highlyscalablesystems.com/3597/hadoop-installation-tutorial-hadoop-2-x/

**Cloudera**, Hadoop Default Ports Quick Reference
Internet, consulté le 17 juillet 2015.
http://blog.cloudera.com/blog/2009/08/hadoop-default-ports-quick-reference/

**Docker Hub Registry**, Repositories of Docker Images
Internet, consulté le 22 juin 2015.
https://registry.hub.docker.com/

**Luciano Resende**, Building a Yarn cluster using Docker.io containers
Internet, consulté le 28 juin 2015.
http://lresende.blogspot.ca/2015/01/building-yarn-cluster-using-dockerio.html

**Tutorials point**, Reading Data using HBase Shell
Internet, consulté le 10 juillet 2015.
http://www.tutorialspoint.com/hbase/hbase_read_data.htm

**Core Dump**, HBase Client Example
Internet, consulté le 25 juillet 2015.
http://cook.coredump.me/post/19672191046/hbase-client-example

**Amandeep Khurana**, Introduction to HBase Schema Design
Internet, consulté le 15 juillet 2015.
https://0b4af6cdc2f0c5998459-c0245c5c937c5dedcca3f1764ecc9b2f.ssl.cf2.rackcdn.com/9353-login1210_khurana.pdf

**For Dummies**, Transitioning from an RDBMS model to HBase
Internet, consulté le 15 juillet 2015.
http://www.dummies.com/how-to/content/transitioning-from-an-rdbms-model-to-hbase.html

**GitHub**, mikefaille
Internet, consulté le 10 juillet 2015.
https://github.com/mikefaille

**Apache Hadoop**, Welcome to Apache Hadoop!
Internet, consulté le 13 mai 2015.
http://hadoop.apache.org/

# ANNEXE I

# COMPTES RENDUS HEBDOMADAIRES

## Mercredi 13 mai 2015

### Ce qui a été fait :

- Lecture du PFE de François Langelier
- Rencontre initiale avec Alain et David pour préciser le projet
- Téléchargement d'Ubuntu 14.04 et configuration d'une VM sur mon laptop pour reproduire la configuration d'un futur étudiant
- Installation de Docker sur ma VM
- Inspection des *repositories* du GitHub
- Récupération et installation des Mac mini chez moi.
- Remplissage de la fiche de renseignement
- Lecture sur les technologies qui seront vues en SYS870

### Éléments bloquants :

- Les Mac mini 3-4-5 avaient un problème au démarrage après le GRUB. Ils affichaient « loading initial ramdisk ».
    - Après avoir communiqué avec François, j'ai appris que passer par le *Recovery Mode* évite le problème.

### Plan d'action :

- Réalisation de la proposition de projet pour le 15 mai
- Lecture supplémentaire sur les technologies de SYS870
- Tester les Mac mini et leur environnement distribué
- Expérimenter avec Docker.
- Commencer la création d'images Docker avec HBase en priorité

## Mardi 19 mai 2015

### Ce qui a été fait :

- Remise de la proposition de projet
- Lecture sur les technologies du cours SYS870
- Apprivoisement de Docker et ses commandes
- Tests d'installation de HBase en Standalone mode
- Conception d'une première ébauche d'un Dockerfile pour construire une image HBase

### Éléments bloquants :

- Avant de lancer « hbase shell », il faut exécuter le script « start-hbase.sh ». Le RUN de mon Dockerfile ne semble pas exécuter le script.
- En lançant manuellement le script et ensuite « hbase shell », j'ai le message « Unable to load native-hadoop library for your platform... using builtin-java classes where applicable ». Je ne sais pas s'il est grave ou s'il peut être ignoré.
- Je dois revalider quelle est la priorité des différentes images et celles qui sont déjà complètes.

### Plan d'action :

- Réviser le Dockerfile HBase avec David pour voir si je suis dans la bonne direction
- Terminer le DockerFile HBase
- Revérifier avec David quelles sont les images à prioriser
- Commencer le travail sur d'autres Dockerfiles

## Mardi 26 mai 2015

### Ce qui a été fait :

- Rencontre de projet
- Installation et tests de boot2docker sur mon laptop
- Modification et tests du dockerfile CDH de David
- Création d'un compte Docker Hub
- Configuration des Mac mini pour les rendre accessibles de l'externe
- Début de tests sur le cluster mac
- Recherches en lien avec CDH

### Éléments bloquants :

- Il n'y avait pas de serveur DNS configuré pour les Macs.
    - Configuration corrigée pour mettre ceux de Google.
- Docker n'était plus installé sur le Mac 1.
    - Le bon package a été installé.
- J'avais des incertitudes concernant le contenu et le rôle de l'image CDH.
    - Une discussion avec David a permis de clarifier les choses pour le moment.

### Plan d'action :

- Faire un dockerfile pour une image CDH-Hive
- Faire un dockerfile pour une image CDH-Impala
- Tester le cluster Mac
- Ajouter mes choses dont le script HBase sur le GitHub

## Mardi 2 juin 2015

### Ce qui a été fait :

- Tests et recherches pour image Hive et Impala
- Troubleshooting de l'image Hadoop
- Ébauche de Dockerfile Redis
- Ébauche de Dockerfile MongoDB

### Éléments bloquants :

- L'installation manuelle de Hive et Impala est assez longue et complexe avec différentes options. J'ai besoin d'être un peu assisté pour être sûr de ce qu'on veut y faire. Je me demande aussi si ça ne serait pas mieux de prendre l'image de Cloudera afin de pouvoir mettre le temps ailleurs.
- Dans les instructions pour leur installation, il y a plusieurs moments où il faut entrer des commandes dans un shell interactif. Je ne suis pas sûr comment ça peut être intégré dans un Dockerfile.
- Je ne maîtrise pas Hadoop/HDFS et je vais avoir besoin de l'aide de David pour vérifier qu'elle est correcte.

### Plan d'action :

- Faire un dockerfile pour une image CDH-Hive
- Faire un dockerfile pour une image CDH-Impala
- S'assurer que l'image Hadoop est correcte
- Ajouter mes choses sur le GitHub

## Mardi 9 juin 2015

### Ce qui a été fait :

- Lecture sur HDFS pour mieux comprendre les différents types de nodes
- Lecture sur le mount de volumes dans Docker
- Tests avec des mounts
- Tests avec l'image Hadoop

### Éléments bloquants :

- Aucun

### Plan d'action :

- Continuer sur Hadoop
- Coordonner pour la préparation de la machine

## Mardi 16 juin 2015

### Ce qui a été fait :

- Upload du dockerfile Hbase sur GitHub
- Tests avec Hadoop et les nodes HDFS

### Éléments bloquants :

- Mes occupations à l'extérieur de l'école m'ont empêché de beaucoup progresser cette semaine.

### Plan d'action :

- Travailler sur les images manquantes
- Repréciser les objectifs et prochaines étapes

## Mardi 23 juin 2015

### Ce qui a été fait :

- Recherches sur l'installation des composantes CDH
- Travail sur le dockerfile CDH

### Éléments bloquants :

- Je ne suis plus certain s'il y a un problème ou non avec l'image Hadoop
- J'ai besoin d'avoir la liste de toutes les images qui sont voulues.
- Il faut reconfirmer ce qu'on veut faire avec CDH (image manuelle ou fournie par Cloudera).

### Plan d'action :

- Rencontre pour repréciser les objectifs et prochaines étapes

## Mardi 30 juin 2015

### Ce qui a été fait :

- Lecture sur la nouvelle version de Docker (1.7) sortie
- Tests avec Docker 1.7
- Lecture à propos du dataset GSOD
- Contact avec Patrice pour la mise en place de la VM
- Récupération des données GSOD 2014 et merge des données en un seul fichier
- Conversion des données GSOD au format CSV
- Tests d'importation d'un CSV dans HBase

### Éléments bloquants :

- Pour l'instant, j'ai une erreur à l'importation en utilisant l'outil importtsv. Le Resource Manager de YARN ne démarre pas et cause l'erreur. Cela semble être lié au fichier capacity-scheduler.xml, mais c'est à confirmer.
- Patrice est absent en ce moment, mais il sera de retour le 2 juillet.

### Plan d'action :

- Continuer de travailler sur l'importation du dataset dans HBase
- Prendre des nouvelles de Patrice pour la VM à son retour

## Mardi 7 juillet 2015

### Ce qui a été fait :

- Rencontre avec David pour mieux déterminer ce qui sera fait avec le dataset GSOD
- Création du data dictionary pour les données GSOD
- Vérifications sur l'installation de R dans Ubuntu
- Troubleshooting de YARN et lecture plus approfondie
- Installation de Hadoop à partir de zéro sans le dockerfile afin d'identifier les différences de configuration avec notre image
- Après plusieurs essais et tests, les services de YARN démarrent finalement bien autant sur un single node que sur un cluster. Il y a une modification à faire à notre dockerfile Hadoop actuel pour qu'un fichier de configuration additionnel soit ajouté dans le conf dir.
- Tests d'importation d'un CSV dans HBase
- Relance à Patrice pour la VM et ajout de l'installation de R à la demande

### Éléments bloquants :

- Les services YARN démarrent et la job d'importation dans HBase débute (state : accepted), mais elle ne semble pas réellement travailler (FinalStatus : undefined). Son statut reste indéfiniment le même. Je fais un test avec un fichier de 20 lignes, donc je sais que ce n'est pas lié à un long temps de traitement. Je dois donc encore travailler sur YARN.

### Plan d'action :

- Faire fonctionner l'importation des données GSOD dans HBase
- Avoir accès à la machine
- Commencer à ajouter des données manuellement dans la table HBase si le problème dure encore. Cela permettra au moins tester des premières requêtes.

## Mardi 14 juillet 2015

### Ce qui a été fait :

- Importation des données sur les mesures GSOD 2014 dans HBase réussie dans un environnement Hadoop/YARN standalone
- Importation des données sur les stations GSOD dans HBase
- Peaufinage de la méthode de formatage des données pour les préparer l'importation et création d'une documentation résumant toutes les étapes du téléchargement à l'arrivée dans HBase
- Accès à la VM faite par Patrice et continuation du travail sur celle-ci
- Tests d'importation et de jobs YARN sur un cluster Hadoop (une machine hbase/namenode/resource manager et une datanode/node manager). L'importation vient tout juste de réussir pour la première fois.

### Éléments bloquants :

- Je ne suis pas root sur la machine.
- Je dois vérifier quelle est l'architecture idéale que nous voulons pour un cluster (nombre de containers, répartition des rôles).

### Plan d'action :

- Faire une demande pour quelques ajustements sur la machine (accès root et Docker 1.7 plutôt que 1.3)
- Commencer à faire des queries dans HBase
- Valider avec David que les deux tables HBase ont une bonne structure (familles, row keys, etc)
- Déterminer les requêtes qui pourraient être demandées pour le labo
- Faire fonctionner l'importation et les services ont demandé plusieurs manipulations manuelles après la création à partir des images Docker. Je dois revalider toutes les étapes à faire et possiblement mettre à jour les dockerfiles.
- Retester le cluster avec la vraie architecture voulue

## 21 juillet 2015

### Ce qui a été fait :

- Revue et refonte de l'image HBase qui est maintenant complètement standalone. Elle est simplifiée, hérite de l'image Java et n'a plus besoin que son service soit lancé manuellement. L'importation des données via une job MapReduce peut aussi se faire à partir du filesystem local. La mise à jour est sur le GitHub.
- Début d'une image Sqoop.
- Tests pour s'assurer d'avoir une persistance des données autant avec HBase que Hadoop/HDFS en cas de perte d'un container. L'image Hadoop a été mise à jour et les

instructions vont l'être très bientôt. La persistance fonctionne en utilisant un bon mount. Les tables HBase restent présentes et les fichiers dans HDFS aussi en cas de problème.
- Adaptation d'un script Bash pour récupérer les données GSOD. Une fois les scripts sur le serveur, il suffit de lancer le script maître en spécifiant l'année de début et de fin. Le résultat est un fichier rassemblant toutes données et formaté. Il sera mis sur un repo privé du GitHub très bientôt.
- Tests pour l'accès à l'interface graphique web de Hadoop à distance. En publiant les bons ports pour chaque type de node, il est possible d'accéder à la page web de chaque service. Je peux faire un browse du HDFS qui est sur le container de la VM de l'ÉTS et télécharger ses fichiers à partir du navigateur web de mon laptop. Cette dernière fonctionnalité demande par contre de modifier son host file pour que les hostnames des nodes HDFS pointent sur l'IP du host Docker.

### Éléments bloquants :

- L'image Sqoop n'est pas pour l'instant fonctionnelle. Il y avait un souci que je n'ai pas encore regardé en profondeur étant donné que ce n'était pas la priorité.
- Je vais probablement devoir revoir la conception de mes tables HBase pour être de mesure de faire les requêtes adéquates pour obtenir les données demandées.
- Importer une seule année dans HBase prend 10 minutes. Il va donc falloir se limiter à quelques-unes plutôt que de tout récupérer.

### Plan d'action :

- Créer des requêtes HBase pour sortir les données par rapport aux questions qui ont été déterminées par David.
- Voir pour l'interface web de HBase.
- Commencer à documenter davantage mes choses
- Vérifier pour l'image Sqoop

## 28 juillet 2015

### Ce qui a été fait :

- Modification des scripts de téléchargement GSOD. Ajout du téléchargement automatique des stations et création d'une documentation.
- Discussions avec Patrice concernant la machine.
- Tests pour le fonctionnement de l'interface web de HBase. Les ports annoncés dans la plupart des documents ont changé autour de la version 0.98 et ont été mis à jour dans l'image.
- Recherche sur la conception des rowkeys dans HBase, les bonnes pratiques et les différents filtres qui peuvent être appliqués.
- Tests de queries dans HBase pour répondre aux questions qui seront posées dans le laboratoire. Essais avec différentes conception pour les tables. Je suis en mesure de répondre à la plupart des questions définies avec David, mais il reste du peaufinage à faire.

- Tests avec l'API Java pour interroger HBase. Les requêtes sont pour l'instant faites directement dans le HBase shell, mais Java pourrait être plus pratique pour certains cas.

## Éléments bloquants :

- Je vais bientôt devoir délaisser temporairement la partie technique pour me concentrer sur la présentation orale et la remise du rapport la semaine prochaine.
- Patrice est en vacances et il restera certains points à clarifier à son retour le 10 août concernant notamment la version de Docker et les permissions des étudiants.

## Plan d'action :

- Travailler sur les queries et l'API Java de HBase.
- Préparation de la présentation orale
- Préparation du rapport

# ANNEXE II

## DOCKERFILE POUR HADOOP

```
# Building the image using my Oracle JDK 7
FROM gelog/java:openjdk7


MAINTAINER Francois Langelier


ENV WGET_VERSION       1.15-1ubuntu1.14.04.1
# Setting HADOOP environment variables
ENV HADOOP_VERSION 2.6.0
ENV HADOOP_INSTALL /usr/local/hadoop
ENV PATH $PATH:$HADOOP_INSTALL/bin
ENV PATH $PATH:$HADOOP_INSTALL/sbin
ENV HADOOP_MAPRED_HOME $HADOOP_INSTALL
ENV HADOOP_COMMON_HOME $HADOOP_INSTALL
ENV HADOOP_HDFS_HOME $HADOOP_INSTALL
ENV HADOOP_COMMON_LIB_NATIVE_DIR $HADOOP_INSTALL/lib/native
ENV YARN_HOME $HADOOP_INSTALL
ENV HADOOP_CONF_DIR $HADOOP_INSTALL/etc/hadoop
# Installing wget
RUN \
    apt-get update && \
    apt-get install -y wget=$WGET_VERSION && \
    rm -rf /var/lib/apt/lists/*
# Installing HADOOP
RUN wget http://archive.apache.org/dist/hadoop/core/hadoop-$HADOOP_VERSION/hadoop-
$HADOOP_VERSION.tar.gz && \
    tar -zxf /hadoop-$HADOOP_VERSION.tar.gz && \
    rm /hadoop-$HADOOP_VERSION.tar.gz && \
    mv hadoop-$HADOOP_VERSION /usr/local/hadoop && \
    mkdir -p /usr/local/hadoop/logs
# Creating symlink for HADOOP configuration files
VOLUME /data
# Copying default HADOOP configuration files
ADD https://raw.githubusercontent.com/GELOG/docker-ubuntu-
hadoop/$HADOOP_VERSION/$HADOOP_VERSION/core-site.xml $HADOOP_CONF_DIR/core-site.xml
ADD https://raw.githubusercontent.com/GELOG/docker-ubuntu-
hadoop/$HADOOP_VERSION/$HADOOP_VERSION/yarn-site.xml $HADOOP_CONF_DIR/yarn-site.xml
ADD https://raw.githubusercontent.com/GELOG/docker-ubuntu-
hadoop/$HADOOP_VERSION/$HADOOP_VERSION/mapred-site.xml $HADOOP_CONF_DIR/mapred-site.xml
ADD https://raw.githubusercontent.com/GELOG/docker-ubuntu-
hadoop/$HADOOP_VERSION/$HADOOP_VERSION/hdfs-site.xml $HADOOP_CONF_DIR/hdfs-site.xml
CMD ["hdfs"]
```

# ANNEXE III

## DOCKERFILE POUR HBASE

```
# Building the image using Oracle JDK 7
FROM gelog/java:openjdk7
MAINTAINER Julien Beliveau
# Setting HBASE environment variables
ENV HBASE_VERSION 1.1.1
ENV HBASE_HOME /usr/local/hbase
ENV PATH $PATH:$HBASE_HOME/bin
# Installing wget
RUN \
    apt-get update && \
    apt-get install -y wget && \
    rm -rf /var/lib/apt/lists/*
# Installing HBase
RUN    wget https://www.apache.org/dist/hbase/$HBASE_VERSION/hbase-$HBASE_VERSION-
bin.tar.gz && \
       tar -xf hbase-$HBASE_VERSION-bin.tar.gz && \
       rm hbase-$HBASE_VERSION-bin.tar.gz && \
       mv hbase-$HBASE_VERSION /usr/local/hbase

# Mounting the HBase data folder and exposing the ports
VOLUME /data
EXPOSE 16000 16010 16020 16030 16100
# Editing the HBase configuration file to use the local filesystem
ADD https://raw.githubusercontent.com/GELOG/docker-ubuntu-hbase/master/conf/hbase-site.xml
$HBASE_HOME/conf/hbase-site.xml
# Starting HBase
CMD $HBASE_HOME/bin/hbase master start
```

## ANNEXE IV

## README.MD DES DOCKERFILES

### Hadoop

```
# How to use this image?
### Formating the namenode (only do it once)
        docker run -d -h hdfs-namenode -v /hostdirectory/docker-
volumes/hdfs-namenode:/data gelog/hadoop:2.6.0 hdfs namenode -format
### Starting the namenode
        docker run -d --name hdfs-namenode -h hdfs-namenode -p 9000:9000 -p
50070:50070 -v /hostdirectory/docker-volumes/hdfs-namenode:/data
gelog/hadoop:2.6.0 hdfs namenode
### Starting a secondary namenode
        docker run -d --name hdfs-secondarynamenode -h hdfs-
secondarynamenode -p 50090:50090 -v /hostdirectory/docker-volumes/hdfs-
secondarynamenode:/data --link=hdfs-namenode:hdfs-namenode
gelog/hadoop:2.6.0 hdfs secondarynamenode
### Starting a datanode
        docker run -d --name hdfs-datanode1 -h hdfs-datanode1 -p
50075:50075 -v /hostdirectory/docker-volumes/hdfs-datanode:/data --
link=hdfs-namenode:hdfs-namenode --link=hdfs-secondarynamenode:hdfs-
secondarynamenode gelog/hadoop:2.6.0 hdfs datanode
### Accessing the web interfaces
        http://hostIP:50070 (namenode)
        http://hostIP:50090 (secondary namenode)
        http://hostIP:50075 (datanode)
```

### HBase

```
# docker-ubuntu-hbase
Dockerfile for running HBase on Ubuntu

# What is HBase?
Apache HBase is an open-source, distributed, versioned, non-relational
database modeled after Google's Bigtable: A Distributed Storage System for
Structured Data by Chang et al.

[https://hbase.apache.org/](https://hbase.apache.org/)

# Base Docker image
* [gelog/java:openjdk7](https://registry.hub.docker.com/u/gelog/java/)

# How to use this image?
### Starting the container
        docker run -d --name hbase-master -h hbase-master -p 16010:16010 -v
/hostdirectory/docker-volumes/hbase-master:/data gelog/hbase

### Accessing the web interface
        http://hostIP:16010
```

# ANNEXE V

## DICTIONNAIRE DE DONNÉES GSOD

| Nom | Description | Type | Exemple |
|---|---|---|---|
| USAF | L'identifiant de la station. Le premier charactère peut être une lettre. | string | A07810 |
| WBAN | Le numéro donné par la Weather Bureau Air Force Navy | int | 99999 |
| NAME | Nom | string | KING MOUNTAIN |
| CTRY | Pays | string | US |
| ST | L'état de la station. Seulement présent pour les États-Unis | string | OR |
| LAT | Latitude en degrés | float | +78.933 |
| LON | Longitude en degrés | float | +028.900 |
| ELEV | Élévation en mètres | float | +0020.0 |
| BEGIN | Date de début de la prise de mesures | date | 20050115 |
| END | Date de fin de la prise de mesures si applicable | date | 20111128 |

| Nom | Description | Type | Exemple |
|---|---|---|---|
| STN--- | L'identifiant de la station | int | 713480 |
| WBAN | Le numéro donné par la Weather Bureau Air Force Navy | int | 99999 |
| DATE | La date de la prise de mesures | date | 20140117 |
| TEMP | La température moyenne de la journée en Fahrenheit | float | 23.3 |
| Count | Le nombre de mesures pour le calcul de la température moyenne | int | 24 |
| DEWP | Le point de rosée moyen de la journée en Fahrenheit | float | 15.9 |
| Count | Le nombre de mesures pour le calcul du point de rosée moyen | int | 24 |
| SLP | La pression moyenne au niveau de la mer de la journée en millibar | float | 1024.5 |
| Count | Le nombre de mesures pour le calcul de la pression moyenne au niveau de la mer | int | 24 |
| STP | La pression moyenne à la station de la journée en millibar | float | 1019.7 |
| Count | Le nombre de mesures pour le calcul de la pression moyenne à la station | int | 24 |
| VISIB | La visibilité moyenne de la journée en miles | float | 31.1 |
| Count | Le nombre de mesures pour le calcul de la visibilité moyenne | int | 7 |
| WDSP | La vitesse moyenne du vent de la journée en knots | float | 3.6 |
| Count | Le nombre de mesures pour le calcul de la vitesse moyenne du vent | int | 24 |
| MXSPD | La vitesse maximale du vent de la journrée en knots | float | 7.8 |
| GUST | La vitesse maxime de rafale de vent de la journée en knots | float | 999.9 |
| MAX | La température maximale de la journée en Fahrenheit | float | 30.7 |

| | | | |
|---|---|---|---|
| Flag | Un champ vide indique que la température maximale a été prise à partir des données explicites. Un * indique qu'elle a été prise à partir des données horaires. | char | * |
| MIN | La température minimale de la journée en Fahrenheit | float | 23.2 |
| Flag | Un champ vide indique que la température minimale a été prise à partir des données explicites. Un * indique qu'elle a été prise à partir des données horaires. | char | * |
| PRCP | Le total des précipations de la journée en pouces incluant la pluie et la neige fondue | float | 0.04 |
| Flag | A = 1 rapport de précipitation pour 6 heures<br>B = Résumé de 2 rapports de précipitation pour 6 heures<br>C = Résumé de 3 rapports de précipitation pour 6 heures<br>D = Résumé de 4 rapports de précipitation pour 6 heures<br>E = 1 rapport de précipitation pour 12 heures<br>F = Résumé de 2 rapports de précipitation pour 12 heures<br>G = 1 rapport de précipitation pour 24 heures<br>H = La station n'a signalé aucune précipitation pour la journée, mais elle en a signalé au moins une dans ses observations horaires.<br>I = La station n'a signalé aucune précipitation pour la journée et aucune occurence dans ses observations horaires. | char | G |
| SNDP | La profondeur de la neige en pouces | float | 6.7 |
| FRSHTT | Indique s'il y a eu une occurrence (1) ou non (0) de certains événements. Chaque chiffre correspond de gauche à droite à: brouillard, pluie/bruine, neige, grêle, tonerre, tornade/entonnoir nuageux. | int | 110000 |

# ANNEXE VI

## SCRIPTS DE RÉCUPÉRATION DES DONNÉES GSOD

### gsod_master_script.sh

```
#!/bin/bash

# Get the stations
./gsod_dl_stations.sh


# Get the data time range from the user
firstYear=$1
lastYear=$2


# Download the data schema that will be used to convert to CSV
wget https://raw.githubusercontent.com/tothebeat/noaa-gsod-data-
munging/master/gsod_schema.csv -O gsod_data_schema.csv


./gsod_dl_all_data.sh $firstYear $lastYear
./gsod_format_all_data.sh $firstYear $lastYear


# Stack every .op year file into one
echo ==============================
echo Merging every year together
echo ==============================
cat *.op > gsod_data_"$firstYear"_to_"$lastYear".op
```

## gsod_dl_stations.sh

```
#!/bin/bash

echo ================================
echo Downloading station info
echo ================================
# Get the file
wget ftp://ftp.ncdc.noaa.gov/pub/data/gsod/isd-history.csv


# Remove the first line
sed 1d isd-history.csv > isd-history-temp.csv


# Remove the double quotes
cat isd-history-temp.csv | tr -d '"' > gsod_stations.csv


# Remove the temp file
rm isd-history.csv isd-history-temp.csv
```

## gsod_dl_all_data.sh

```
#!/bin/bash

# Inspired by: https://github.com/tothebeat/noaa-gsod-data-munging/


firstYear=$1
lastYear=$2


for currentYear in `seq $firstYear $lastYear`; do
    ./gsod_dl_data_year_XXXX.sh $currentYear
done
```

## gsod_dl_data_year_XXXX.sh

```
#!/bin/bash

# Inspired by: https://github.com/tothebeat/noaa-gsod-data-munging/


# Create a directory for the year
year=$1
if [ ! -d "$year" ]; then
    echo "Creating a directory."
    mkdir -p $year
fi


# Download the data (if we don't already have it)
if [ ! -f "$year/gsod_$year.tar" ]; then
       echo ==============================
       echo Downloading $year
       echo ==============================
    wget ftp://ftp.ncdc.noaa.gov/pub/data/gsod/$year/gsod_$year.tar -O
$year/gsod_$year.tar
fi


# Unzip the data
if [ -f "$year/gsod_$year.tar" ]; then
    echo "Unzipping the downloaded data."
    tar -xf $year/gsod_$year.tar -C $year/
    rm $year/gsod_$year.tar
    for filename in `ls $year/*.gz`; do
        gunzip $filename
    done
fi
```

## gsod_format_all_data.sh

```
#!/bin/bash

# Inspired by: https://github.com/tothebeat/noaa-gsod-data-munging/


firstYear=$1
lastYear=$2


for currentYear in `seq $firstYear $lastYear`; do
    ./gsod_format_data_year_XXXX.sh $currentYear
done
```

## gsod_format_data_year_XXXX.sh

```
#!/bin/bash
# Inspired by: https://github.com/tothebeat/noaa-gsod-data-munging/


year=$1


echo ================================
echo Formatting $year
echo ================================

# Strip the first line of each of the .op files
for filename in `ls $year/*.op`; do
    tail -n +2 $filename > $filename.header_stripped
    mv $filename.header_stripped $filename
done


# Stack every .op measure file into one for the year
cat $year/*.op > gsod_data_"$year".op


rm -rf $year
```

# ANNEXE VII

## DOCUMENTATION SUR LES SCRIPTS GSOD

To get the data:

- Download the six GSOD bash scripts somewhere on the host.
- Make sure that you can execute them.

  ***chmod +x gsod*.sh***
- Run the GSOD master script specifying the first and last year to download.

  ***./gsod_master_script.sh 2000 2014***
- There will be an .op file for every year (e.g. "gsod_data_2014.op") and one with everything merged (e.g. "gsod_data_2000_to_2014.op").
- There will also be a gsod_stations.csv file with all the data related to the stations.
- The gsod_data_schema.csv file will be used to convert the op file to csv.

To convert the .op files to CSV:

- Edit gsod_data_schema.csv to have the columns in the wanted order
- Make sure that csvkit is installed.

  ***sudo apt-get update***

  ***sudo apt-get install python-setuptools***

  ***sudo easy_install pip***

  ***sudo pip install csvkit***
- Run the following command for one year or everything:

  ***in2csv -s gsod_data_schema.csv gsod_data_2010.op > gsod_data_2010.csv***

  or

  ***in2csv -s gsod_data_schema.csv gsod_data_2010_to_2014.op > gsod_data_2000_to_2014.csv***
- Remove the first line of the CSV file

  **sed 1d *filename* > *newFilename*.csv**

# ANNEXE VIII

## QUESTIONS CANDIDATES POUR LE LABORATOIRE

Note : Les dates pourront être changées

1) Quelles sont toutes les données météo de la station SORKJOSEN pour le 25 décembre 2014? À faire via le shell hbase.
2) Quelle a été la température minimale, moyenne et maximale au Rhode Island entre le 15 mai et le 14 avril 2002?
3) Nommez les stations où il a fait en moyenne moins de 5 F le 2 février 2008
4) Dans la région de Montréal ([45.405] à [45.705], [-73.47] à [-73.98]), nommez les occurrences (station et date) où il y a eu de la grêle entre 2000 et 2014.
5) Quelles sont les températures en Celsius de la station 008414 en 2014?

# ANNEXE IX

## COMMANDES ET PROGRAMMES JAVA RÉPONDANT AUX QUESTIONS

### Question 1

La table gsod_data_stn_date utilise une rowkey *stationID_YYYYMMDD.*

Exécuter la commande suivante dans le HBase shell :

scan 'gsod_stations', {FILTER => SingleColumnValueFilter.new(Bytes.toBytes('d'), Bytes.toBytes('name'), CompareFilter::CompareOp.valueOf('EQUAL'), BinaryComparator.new(Bytes.toBytes('SORKJOSEN')))}

Cela nous donnera le stationID de la station SORKJOSEN. Exécuter ensuite la requête suivante en mettant le bon stationID :

get 'gsod_data_stn_date', '*stationID*_20141225'

### Question 2

```
/**
   HBase queries to answer the following question:
       "Quelle a ete la temperature minimale, moyenne et maximale au Rhode Island entre le
15 mai et le 14 avril 2002?"

   The queries are based on the following rowkey design:
       gsod stations table: "county_state_stationID"
       gsod data table: "stationID_YYYYMMDD"

   @author Julien Beliveau
**/
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.client.Scan;
import org.apache.hadoop.hbase.client.HTable;
import org.apache.hadoop.hbase.client.Result;
import org.apache.hadoop.hbase.client.ResultScanner;
import org.apache.hadoop.hbase.util.Bytes;


public class HbaseQueryQ2{
```

```
   public static void main(String[] args) throws IOException, Exception{

         String state = "US_RI";
         String beginDate = "20020415";
         String endDate = "20020515";
         double lowestMinTemp = 9999;
         double highestMaxTemp = -9999;
         double totalAvTemp = 0;
         int avTempCount = 0;

         int stateAscii = ((int) state.charAt(state.length()-1)) + 1;
         char nextStateLastChar = (char)stateAscii;
         String nextState = state.substring(0,state.length()-1) + nextStateLastChar;

      // Instantiating Configuration class
      Configuration config = HBaseConfiguration.create();


      // Instantiating HTable class
      HTable tableStations = new HTable(config, "gsod_stations_2");
      HTable tableData = new HTable(config, "gsod_data_stn_date");


         // Scan settings for the station table
         Scan scanStations = new Scan(Bytes.toBytes(state), Bytes.toBytes(nextState));
         ResultScanner scannerStations = tableStations.getScanner(scanStations);



         // Going through the station table
         for (Result resultStations : scannerStations) {

               String rowKeyStation = Bytes.toString(resultStations.getRow());

               // From the station rowkey, we can get the station ID
               String stationID = rowKeyStation.substring(rowKeyStation.lastIndexOf('_') +
1);

               String firstStation = new String(stationID + "_" + beginDate);
               String lastStation = new String(stationID + "_" + endDate);


               // Scan settings for the data table
               Scan scanData = new Scan(Bytes.toBytes(firstStation),
Bytes.toBytes(lastStation));
           scanData.addColumn(Bytes.toBytes("d"), Bytes.toBytes("min"));
               scanData.addColumn(Bytes.toBytes("d"), Bytes.toBytes("max"));
               scanData.addColumn(Bytes.toBytes("d"), Bytes.toBytes("temp"));
               ResultScanner scannerData = tableData.getScanner(scanData);


            // Going through the data table
               for (Result resultData : scannerData) {

                     // Min temperature
                     String minTemp = new
String(resultData.getValue(Bytes.toBytes("d"),Bytes.toBytes("min")));
```

```
                        double currentMinTemp = Double.parseDouble(minTemp);
                        if (currentMinTemp < lowestMinTemp)
                                lowestMinTemp = currentMinTemp;

                        // Max temperature
                        String maxTemp = new
String(resultData.getValue(Bytes.toBytes("d"),Bytes.toBytes("max")));
                        double currentMaxTemp = Double.parseDouble(maxTemp);
                        if (currentMaxTemp > highestMaxTemp)
                                highestMaxTemp = currentMaxTemp;

                        // Mean temperature
                        String avTemp = new
String(resultData.getValue(Bytes.toBytes("d"),Bytes.toBytes("temp")));
                        double currentAvTemp = Double.parseDouble(avTemp);
                        totalAvTemp += currentAvTemp;
                        avTempCount++;
                }
        }

        // Calculating the final mean temperature
        double finalAvTemp = totalAvTemp / avTempCount;

        // Displaying the final results

System.out.println("================================================================");
        System.out.println("Between " + beginDate + " and " + endDate);
        System.out.println("Min temp: " + Double.toString(lowestMinTemp) + " F");
        System.out.println("Max temp: " + Double.toString(highestMaxTemp) + " F");
        System.out.println("Average temp: " + Double.toString(finalAvTemp) + " F");

System.out.println("================================================================");


        // Ending the table connections
        tableStations.close();
        tableData.close();
   }
}
```

## Question 3

```
/**
   HBase queries to answer the following question:
       "Nommez les stations ou il a fait en moyenne moins de 5 F le 2 fevrier 2008"

   The queries are based on the following rowkey design:
       gsod stations table: "stationID_wban"
       gsod data table: "YYYYMMDD_stationID"

   @author Julien Beliveau
**/
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.client.Get;
import org.apache.hadoop.hbase.client.Scan;
import org.apache.hadoop.hbase.client.HTable;
import org.apache.hadoop.hbase.client.Result;
import org.apache.hadoop.hbase.client.ResultScanner;
import org.apache.hadoop.hbase.util.Bytes;


public class HbaseQueryQ3{

   public static void main(String[] args) throws IOException, Exception{

         String date =  "20080202";

         int dateAscii = ((int) date.charAt(date.length()-1)) + 1;
         char nextDateLastChar = (char)dateAscii;
         String nextDate = date.substring(0,date.length()-1) + nextDateLastChar;

      // Instantiating Configuration class
      Configuration config = HBaseConfiguration.create();


      // Instantiating HTable class
      HTable tableStations = new HTable(config, "gsod_stations");
      HTable tableData = new HTable(config, "gsod_data_date_stn");


         // Scan settings for the data table
         Scan scanData = new Scan(Bytes.toBytes(date), Bytes.toBytes(nextDate));
         scanData.addColumn(Bytes.toBytes("d"), Bytes.toBytes("temp"));
         scanData.addColumn(Bytes.toBytes("d"), Bytes.toBytes("wban"));
         ResultScanner scannerData = tableData.getScanner(scanData);



System.out.println("================================================================");


         // Going through the data table
         for (Result resultData : scannerData) {
              String temp = new
```

```
String(resultData.getValue(Bytes.toBytes("d"),Bytes.toBytes("temp")));
                double currentTemp = Double.parseDouble(temp);

                if (currentTemp < 5){
                        String rowKeyData = Bytes.toString(resultData.getRow());

                        // From the data rowkey, we can get the station ID
                        String stationID = rowKeyData.substring(rowKeyData.lastIndexOf('_') +
1);

                        String wban = new
String(resultData.getValue(Bytes.toBytes("d"),Bytes.toBytes("wban")));

                        // Having the station ID and the WBAN, we have all the info to do a
Get on the station table for a specific rowkey
                        String rowKeyStation = stationID + "_" + wban;
                        Get getStation = new Get(Bytes.toBytes(rowKeyStation));
                        Result resultStation  = tableStations.get(getStation);

                        // We only want the names
                        byte [] value =
resultStation.getValue(Bytes.toBytes("d"),Bytes.toBytes("name"));
                        String stationName = Bytes.toString(value);

                        // Displaying the final answer
                        System.out.println(stationName + ": " + temp + " F");
                }
        }


System.out.println("================================================================");


         // Ending the table connections
         tableStations.close();
         tableData.close();
   }
}
```

## Question 4

```
/**
   HBase queries to answer the following question:
       "Dans la région de Montréal ([45.405] à [45.705], [-73.47] à [-73.98]), nommez les
       occurrences (station et date) où il y a eu de la grêle entre 2000 et 2014."

   The queries are based on the following rowkey design:
       gsod stations table: "county_state_stationID"
       gsod data table: "stationID_YYYYMMDD"

   @author Julien Beliveau
**/
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.client.Scan;
import org.apache.hadoop.hbase.client.HTable;
import org.apache.hadoop.hbase.client.Result;
import org.apache.hadoop.hbase.client.ResultScanner;
import org.apache.hadoop.hbase.util.Bytes;


public class HbaseQueryQ4{

   public static void main(String[] args) throws IOException, Exception{

         String country = "CA";
         String beginDate = "2000";
         String endDate = "2014";

         int countryAscii = ((int) country.charAt(country.length()-1)) + 1;
         char nextCountryLastChar = (char)countryAscii;
         String nextCountry = country.substring(0,country.length()-1) +
nextCountryLastChar;

      // Instantiating Configuration class
      Configuration config = HBaseConfiguration.create();


      // Instantiating HTable class
      HTable tableStations = new HTable(config, "gsod_stations_2");
      HTable tableData = new HTable(config, "gsod_data_stn_date");


         // Scan settings for the station table
         Scan scanStations = new Scan(Bytes.toBytes(country), Bytes.toBytes(nextCountry));
         scanStations.addColumn(Bytes.toBytes("d"), Bytes.toBytes("lat"));
         scanStations.addColumn(Bytes.toBytes("d"), Bytes.toBytes("lon"));
         ResultScanner scannerStations = tableStations.getScanner(scanStations);



System.out.println("================================================================");
```

```
        // Going through the station table
        for (Result resultStations : scannerStations) {

             String rowKeyStation = Bytes.toString(resultStations.getRow());
             String latitude = new
String(resultStations.getValue(Bytes.toBytes("d"),Bytes.toBytes("lat")));
             String longitude = new
String(resultStations.getValue(Bytes.toBytes("d"),Bytes.toBytes("lon")));

             if (!latitude.equals("") && !longitude.equals("")){
                    double currentLatitude = Double.parseDouble(latitude);
                    double currentLongitude = Double.parseDouble(longitude);

                    if (currentLatitude >= 39.405 && currentLatitude <= 46.705 &&
currentLongitude >= -90.125 && currentLongitude <= -70.575){
                           // From the station rowkey, we can get the station ID
                           String stationID =
rowKeyStation.substring(rowKeyStation.lastIndexOf('_') + 1);

                           int dateAscii = ((int) endDate.charAt(endDate.length()-1)) +
1;
                           char nextDateLastChar = (char)dateAscii;
                           String nextDate = endDate.substring(0,endDate.length()-1) +
nextDateLastChar;

                           // Scan settings for the data table
                           Scan scanData = new Scan(Bytes.toBytes(stationID + "_" +
beginDate), Bytes.toBytes(stationID + "_" + nextDate));
                           scanData.addColumn(Bytes.toBytes("d"), Bytes.toBytes("hail"));
                           ResultScanner scannerData = tableData.getScanner(scanData);

                           for (Result resultData : scannerData) {

                                  String rowKeyData =
Bytes.toString(resultData.getRow());

                                  // hail
                                  String hail = new
String(resultData.getValue(Bytes.toBytes("d"),Bytes.toBytes("hail")));

                                  if ((hail).equals("1"))
                                         System.out.println("Hail for " + rowKeyData);
                           }
                    }
             }
        }


System.out.println("================================================================");


        // Ending the table connections
        tableStations.close();
        tableData.close();
   }
}
```

## Question 5

```
/**
   HBase queries to answer the following question:
       "Quelles sont les temperatures en Celsius de la station 008414 en 2014?"

   The queries are based on the following rowkey design:
       gsod data table: "stationID_YYYYMMDD"

   @author Julien Beliveau
**/
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.client.Scan;
import org.apache.hadoop.hbase.client.HTable;
import org.apache.hadoop.hbase.client.Result;
import org.apache.hadoop.hbase.client.ResultScanner;
import org.apache.hadoop.hbase.util.Bytes;


public class HbaseQueryQ5{

   public static void main(String[] args) throws IOException, Exception{

         String year =  "2014";
         String stationID = "008414";

         int yearAscii = ((int) year.charAt(year.length()-1)) + 1;
         char nextYearLastChar = (char)yearAscii;
         String nextYear = year.substring(0,year.length()-1) + nextYearLastChar;

      // Instantiating Configuration class
      Configuration config = HBaseConfiguration.create();


      // Instantiating HTable class
      HTable tableData = new HTable(config, "gsod_data_stn_date");

         // Scan settings for the data table
         Scan scanData = new Scan(Bytes.toBytes(stationID + "_" + year),
Bytes.toBytes(stationID + "_" + nextYear));
         scanData.addColumn(Bytes.toBytes("d"), Bytes.toBytes("temp"));
         ResultScanner scannerData = tableData.getScanner(scanData);



System.out.println("================================================================");


         // Going through the data table
         for (Result resultData : scannerData) {

               String rowKeyData = Bytes.toString(resultData.getRow());

               String temp = new
```

```
String(resultData.getValue(Bytes.toBytes("d"),Bytes.toBytes("temp")));
                double currentTempF = Double.parseDouble(temp);
                double currentTempC = Math.round((((currentTempF - 32)*5)/9) * 10.0) / 10.0;


                System.out.println(rowKeyData + ": " + Double.toString(currentTempC) + "
C");
        }



System.out.println("=================================================================");


        // Ending the table connections
        tableData.close();
    }
}
```