ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

par
MOHAMAD AWADA

RAPPORT DE PROJET PRÉSENTÉ À
L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

BIGDATA APPLIQUÉ AU SÉQUENÇAGE GÉNÉTIQUE

MONTRÉAL, LE 28 JUIN 2017

PRÉSENTATION DU JURY

CE RAPPORT DE PROJET A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE :

Professeur Alain April, directeur de projet
Département de génie logiciel et TI à l'École de technologie supérieure

Professeur Abdelaoued Gherbi, jury
Département de génie logiciel et TI à l'École de technologie supérieure

REMERCIEMENTS

Je tiens à remercier toutes les personnes qui ont contribué au succès de mon projet.

Tout d'abord, je tiens à remercier le professeur Alain April pour ce privilège, sa confiance en moi, de ses conseils au niveau personnel et professionnel et son encadrement tout au long du projet.

Je remercie aussi David Lauzon pour son encadrement durant le projet et ses conseilles techniques qui ont était cruciales au sucées du projet.

Je tiens aussi à remercier Beatriz Kanzki pour son support et son aide en ce qui concerne la compréhension du domaine.

Enfin, je remercie mes parents Mariam et Kamel Awada, mon épouse Zahraa Harb et mon petit ange Lynn pour leur soutien, leur encouragement et les sacrifices vécus tout au long de mon cheminement.

BIGDATA APPLIQUÉ AU SÉQUENÇAGE GÉNÉTIQUE

MOHAMAD AWADA

RÉSUMÉ

Ce projet de recherche appliquée de 15 crédits en BigData appliqué au séquençage génétique était composé de deux parties. La première partie avait pour but d'explorer les outils d'imputation génétique et de les comparer. Cette première phase du projet a eu lieu au sein de l'équipe de recherche du docteur Pavel Hamet au Centre Hospitalier de l'Université de Montréal (CRCHUM). Le but de cette étude vise à identifier les outils utilisés présentement, dans l'industrie, et de synthétiser les caractéristiques de chaque outil. De plus, lors de cette étude, une attention particulière vise l'identification des données et des paramètres utilisés lors d'une imputation génétique. La deuxième partie du projet de recherche appliquée consiste à analyser un logiciel libre, qui utilise la technologie BigData, qui permet au bio-informaticien de facilement répéter une expérience. Ce type de logiciel de traçabilité des expérimentations capture les transformations qui sont effectuées sur les données lors de l'exécution d'une expérience et permet de changer des paramètres pour rejouer l'expérience ultérieurement. En d'autres mots, cette partie du projet s'intéresse à la notion de la provenance. Ce rapport présente la synthèse des travaux effectués pour les deux sujets.

Mots-clés: BigData, imputation génétique, traçabilité, provenance, logiciel libre.

BIGDATA APPLIED TO GENETIC SEQUENCING

MOHAMAD AWADA

ABSTRACT

This 15 cresdits applied research project, in BigData applied to genetic sequencing, had two phases. The first part of the project consisted of comparing genetic imputation tools with the intention of selecting one for future adaptation. This first phase of the project took place within the research team of Dr. Pavel Hamet at the Center Hospitalier de l'Université de Montréal (CRCHUM). This comparative study identified tools currently used in the industry and synthesizes the characteristics of each tool. Another focus of this study aims at identifying data and parameters typically used for genetic imputation. The second part of the project involves analyzing open source software, which is using BigData technology, which allows bioinformatics specialists to repeat an experiment after the fact. This type of experimental traceability software typically captures the transformations that are performed on the data during the execution of an experiment and allows replaying the experiment and change parameters if wanted. In other words, this second part of the project is concerned with the notion of provenance in genetic research. This report presents the synthesis of the work carried out for the two subjects.

Keywords: BigData, genetic imputation, traceability, provenance, free software.

TABLE DES MATIÈRES

Page

INTRODUCTION 1

CHAPITRE 1 REVUE LITTÉRATURE 3
1.1 L'imputation génétique 3
1.2 Méthodologie d'inférence 4
1.3 Les outils d'imputation 6
1.4 Méthodologie d'imputation des outils 8
1.4.1 MACH 8
1.4.2 IMPUTE2 9
1.4.3 fastPHASE 9
1.4.4 BEAGLE 10
1.5 Discussion 11
1.6 Conclusion 13

CHAPITRE 2 MODULE D'IMPUTATION Q(n)GENE 15
2.1 Objectifs 15
2.2 Technologies courantes 15
2.3 Format des fichiers IMPUTE2 16
2.4 Format VCF 17
2.5 Solution ADVANCE-to-ADAM avec IMPUTE2 18
2.6 Solution ADVANCE-to-ADAM avec BEAGLE 19
2.7 Conclusion 20

CHAPITRE 3 OUTIL DE PROVENANCE 21
3.1 Problématique 21
3.2 Objectifs de la solution 21
3.3 Les défis de la solution 23
3.4 Conclusion 23

CHAPITRE 4 ENVIRONNEMENT DE TRAVAIL 25
4.1 Objectifs 25
4.2 ZeppelinExtractor 25
4.3 Zeppelin 25
4.4 Configuration de Zeppelin 26
4.5 Apache HTTP Client 26
4.6 Jackson 26
4.7 PostgreSQL 27
4.8 Apache Spark 27
4.9 Conclusion 27

CHAPITRE 5 CONCEPTION DE LA SOLUTION 29

5.1 Objectifs 29
5.2 Architecture 29
5.3 Implémentation du module ZeppelinExtractor 31
5.4 Les données Zeppelin 35
5.5 Base de données 38
5.6 Conclusion 41

CHAPITRE 6 SYSTHÈSES DU PROJET 42
6.1 Objectifs 42
6.2 Difficultés rencontrées 42
6.3 Leçons apprises 43
6.4 Recommandations 43

CONCLUSION 45

ANNEXE I ADAM-PLINK GWAS DATA MAPPING 47

ANNEXE II INTERFACE UTILISATEUR ZEPPELIN 51

ANNEXE III FICHIER ZEPPELIN-INTERPRETER-SPARK-UBUNTU.LOG 53

ANNEXE IV FICHIER ZEPPELIN-UBUNTU.LOG 54

ANNEXE V WATCHSERVICE 55

ANNEXE VI L'INTERFACE ZEPPELIN INTERPRETER 56

ANNEXE VII INSTALLATION DES OUTILS 63

BIBLIOGRAPHIE 65

LISTE DES TABLEAUX

Page

Tableau 1.1.1 Comparaison des outils d'inférence (Imputation) 11

Tableau 5.1 Exemple de données dans la table « data » 39

LISTE DES FIGURES

Page

Figure 1.1 Architecture des modules d'analyse génétique GWAS 3

Figure 1.2 Scénario d'imputation avec le panneau de référence [16] 5

Figure 1.3 Étude de comparaison des outils d'imputation [9].....7

Figure 1.4 Imputation traditionnelle vs préphasage.....7

Figure 2.1 Format «.gen».....17

Figure 2.2 Format « .sample » 17

Figure 2.3 Format VCF [25] 18

Figure 2.4 Solution ADVANCE-to-ADAM [27]. 19

Figure 2.5 Solution ADVANCE-to-ADAM avec BEAGLE.....20

Figure 3.1 Solution outil de provenance 22

4.1 Architecture des interpréteurs Zeppelin [29]. 28

Figure 5.1 Architecture ZeppelinExtractor et l'interaction des différents modules 30

5.2 Requête HTTP afin d'extraire les carnets existants 32

5.3 Conversion des objets Json en objet Java 32

5.4 Exemple des mots-clés rechercher dans un texte.....33

5.5 Extraction des données utilisées lors de la requête HTTP 33

5.6 Fragment du code source démontrant les informations retirées lors de l'exécution d'un paragraphe.....34

5.7 Insertion de données dans la base de données 35

5.8 Requête qui permet d'identifier les paragraphes existants dans un carnet 36

5.9 Requête qui permet d'extraire les informations d'un paragraphe exécuter 37

5.10 Exemple de fichier journal contenant l'identifiant du paragraphe exécuter.....37

5.11 Exemple de fichier journal contenant le fichier entrant.....37

5.12 Validation de la création de la table « data » 38

5.13 Validation des colonnes dans la table « data » 39

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

ÉTS: École de technologie supérieure

CRCHUM: Centre de Recherche du Centre Hospitalier Universitaire de Montréal

GWAS: Genome-Wide Association Study

SNP: Single-Nucleotide Polymorphism

VCF: Variant Call Format

ADVANCE: Action in diabetes and vascular disease

HTTP: Protocole de transfert hypertexte

LISTE DES DÉFINITIOS

SNP: «...Constituent la forme la plus abondante de variations génétiques dans le génome humain. Ils représentent plus de 90% de toutes les différences entre individus. C'est un type de polymorphisme de l'ADN dans lequel deux chromosomes diffèrent sur un segment donné par une seule paire de base.» [12].

Locus: Un emplacement physique précis et invariable sur un chromosome

Phénoptype: Ensemble des gènes d'un individu

Haplotype: Ensemble de gènes situés sur le même chromosome.

Panneau de référence: Contiens des génotypes typés des personnes de plusieurs régions géographiques, groupe ethnique et pays pour prédire les SNPs manquants dans un échantillon chez un individu. Les panneaux de référence utilisés dans l'industrie sont: 1) HapMap; et 2) 1000 Genomes project.

Homozygote: « se dit d'un gène qui, chez un individu (animal ou végétal), sera représenté par deux allèles (des variantes de ce gène) identiques, sur un même locus. » [19].

Variant Call Format : Un format de fichier texte pour stocker des données de marqueurs et de génotypes

ADAM: «...un ensemble de formats, d'API et d'implémentations de phase de traitement pour les données génomiques. ADAM est entièrement libre sous la licence Apache 2, et est implémenté sur Avro et Parquet pour le stockage de données.» [21].

ADVANCE: Base de donnée utilisée par l'équipe de recherche CRCHUM contenant les données des individus avec leur information génotype.

Marker: «Un marqueur est une mutation qui distingue («marques») une souche. Un marqueur peut être une mutation génique ou un phénotype.» [26].

Spark: Apache Spark est un système de grappe (de l'anglais « cluster ») rapide pour le traitement de l'information à grande échelle. Spark fournit des API de haut niveau en Java, Scala, Python et R.

INTRODUCTION

Le domaine de la bio-informatique est un domaine multidisciplinaire dans lequel les bioinformaticiens utilisent des logiciels, des algorithmes mathématiques et des notions de biologie et de médecine afin d'accomplir leur tâche. De nos jours, la technologie et les logiciels sont des outils indispensables dans les laboratoires de recherche médicale. Même avec la présence de ces nombreuses technologies et la compétence des bioinformaticiens, les centres de recherche en santé et les centres hospitaliers universitaires sont toujours à la recherche de nouvelles technologies afin de réduire le cycle des découvertes. Des efforts croissants sont dévoués à l'analyse génétique. Ils visent à tenter d'identifier très tôt les maladies potentielles, qui pourraient survenir chez un patient, et ainsi que les traitements préventifs possibles. Pour ces centres de recherche et notre société, la prévention est la meilleure approche pour l'amélioration des soins des générations à venir.

Les technologies émergentes du BigData sont prometteuses et pourraient permettre aux bioinformaticiens et à ces centres de recherche d'améliorer les activités d'analyse génétique. Ces technologies pourraient leur permettre d'accomplir les tâches quotidiennes de traitement de quantités massives de données plus rapidement, de traiter un plus grand volume de donnée génétique et clinique lors des analyses et ce à faible coût.

Afin d'introduire ces technologies émergentes aux bioinformaticiens, une équipe ayant cette connaissance et cette expertise pourrait accélérer son utilisation. À cette fin, le professeur Alain April, de l'ÉTS, collabore avec l'équipe de recherche du Dr Pavel Hamet, du CRCHUM, et ses étudiants participent à des projets de recherche appliquée.

Ce rapport de recherche appliquée possède deux parties distinctes. Sa première partie a comme objectif de présenter une synthèse des différents logiciels d'imputation actuellement disponibles et de présenter une étude comparative sommaire avec l'objectif futur : 1) d'en sélectionner un; et 2) de l'adapter aux technologies BigData lors d'un projet de recherche futur. Sa deuxième partie présente initialement la technique ainsi que la technologie utilisée,

en ce qui concerne un logiciel qui pourrait être utilisé ultérieurement dans l’équipe de recherche afin de capturer les transformations qui sont effectuées lors d’une analyse génétique et permettre de changer les paramètres afin de rejouer l'expérience ultérieurement.

CHAPITRE 1

REVUE LITTÉRATURE

1.1 L'imputation génétique

L'imputation génétique est l'inférence des génotypes manquants. C’est-à-dire le processus qui permet de déduire les génotypes qui ne peuvent être typés directement. De plus, l'imputation est une étape critique en ce qui concerne l'analyse des variantes génétiques qui sont contenues dans l’étude Genome-Wide Association Study (GWAS) [13]. Les extrants de ce module sont utilisés par le module de GWAS afin de déterminer si les variantes génétiques augmentent le risque d'une maladie chez un individu. Un autre avantage de l'imputation est le fait de fournir une vue d'ensemble, à haute résolution, d'un signal d'association à travers un locus. La figure suivante, la Figure 1.1, fournit une vue globale du processus de l'analyse génétique du laboratoire du Dr. Pavel Hamet. En outre, cette figure permet de localiser l’emplacement du module d'imputation dans cette chaine de traitements effectués lors d’une analyse génétique.

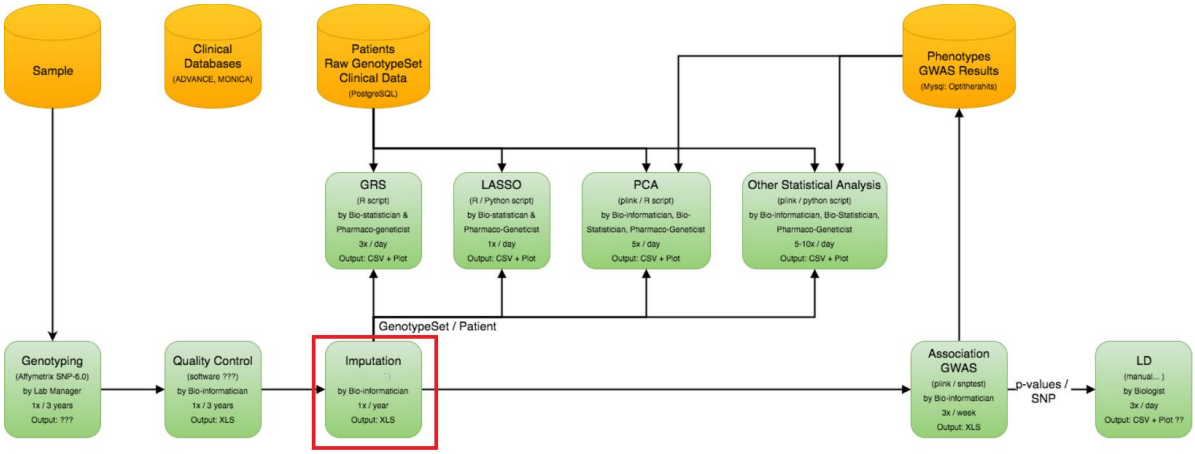


Figure 1.1 Architecture des modules d'analyse génétique GWAS

1.2 Méthodologie d'inférence

Afin d'inférer les génotypes manquants, les bio-informaticiens utilisent des logiciels existant avec des panneaux de référence qui leur permet de déduire les SNP non typés. Les logiciels d'imputation utilisent le modèle de Markov caché (HMM) [2] comme modèle mathématique afin de calculer la probabilité des valeurs inobservées. Le modèle de Markov caché est « un modèle statistique dans lequel le système modélisé est censé être un processus Markovien aux paramètres inconnus. » [2]. Le fonctionnement de cet algorithme est décrit sur le site web du National Center for Biotechnology Information [6].

Un autre algorithme populaire est l'algorithme de grappe haplotye (c.-à-d. Haplotype Clustering Algorithm). Cet algorithme permet le « regroupement des haplotypes dans le but spécifique de traiter la diversité des haplotypes au sein où entrent les populations. L'algorithme permet d'avoir des données manquantes dans les haplotypes et ainsi de réduire de façon appropriée les polymorphismes de nucléotides simples (SNP) qui comportent une plus grande étendue de données manquantes. En identifiant les haplotypes canoniques dans une région génomique, définie comme les formes haplotypiques, dont la plupart des chromosomes sont similaires, l'algorithme mappe chaque chromosome soit à un haplotype canonique unique, soit comme une mosaïque des haplotypes canoniques identifiés.» [11].

Les panneaux de référence ont pour but de fournir des SNP génotypés de plusieurs individus, provenant de différentes régions du monde. En d'autres mots, ces panneaux fournissent des échantillons d'individus de toute provenance et ces échantillons sont utilisés afin d'inférer les SNP non typés de la personne sous analyse. Il faut mentionner ici le fait que les données de ces échantillons ne sont pas toutes utilisées lors de l'analyse. Les données utilisées typiquement sont les échantillons des personnes qui sont de la même région géographique, du même groupe ethnique et du même pays de la personne sous analyse. Ceci a pour but d'augmenter la qualité des génotypes imputés. Par exemple, il sera difficile de trouver des haplotypes correspondants dans un panneau de référence qui contient des échantillons des personnes de descendance européenne si l'étude est effectuée sur une personne de

descendance asiatique.([14], section 7.3) La prochaine figure, la figure 1.2, fournit un scénario typique d'une inférence à partir d'un tableau de référence.

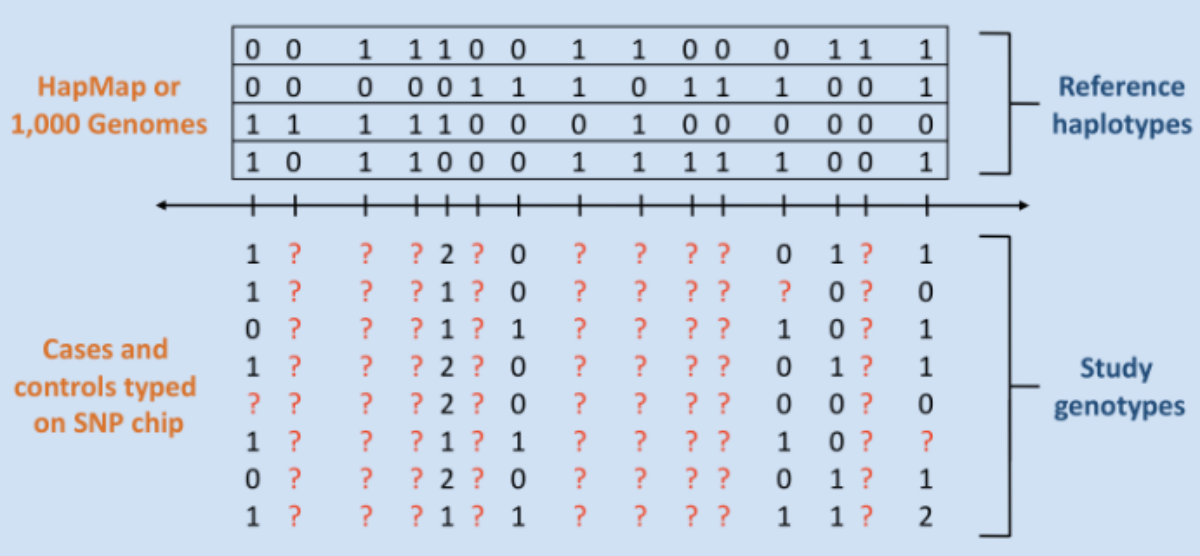


Figure 1.2 Scénario d'imputation avec le panneau de référence [16]

La figure ci-dessus démontre un scénario ou il existe des génotypes non typés (points d'interrogation rouges) d'un ensemble d'individus. Les génotypes non typés sont inférés en utilisant un ensemble de génotypes de référence (par exemple en utilisant les bases de données de référence HapMap ou 1000 Genomes project).

Le première section du tableau de référence est celui du HapMap. Le projet HapMap avait comme but de « cartographier et comprendre les tendances de la diversité génétique commune dans le génome humain afin d'accélérer la recherche des causes génétiques des maladies humaines.» [3]. Ce projet comprend les échantillons d'environ 1 millier de personnes comportant 4 millions de variations.

La deuxième section du tableau de référence est celui du « 1000 Genomes Project ». Ce projet, démarré en janvier 2008, est une recherche internationale qui vise à établir un catalogue de variations génétiques humaines détaillées. [8]. Ce projet inclut plus de 2000 échantillons provenant de 26 pays différents ce qui représente environ 40 millions de variant.

1.3 Les outils d'imputation

Les caractéristiques les plus recherchées lors de l'utilisation d'un logiciel d'imputation génétique sont 1) la précision; et 2) le temps d'exécution d'une imputation génétique ([9], Introduction). Les logiciels présentés dans cette section fournissent des précisions et des performances différentes. Voici quelques facteurs qui peuvent affecter la précision, la performance et la qualité d'une imputation génétique :

- Panneau de référence;
- Modification aux algorithmes;
- Paramètres des entrants;
- Méthodologie d'imputation;
- Les technologies utilisées (c.-à-d. la mémoire vive, le système d'exploitation et le langage de programmation).

Une étude effectuée par l'entreprise Golden Helix [9] a permis de comparer les logiciels les plus populaires. Cette étude compare le temps de calcul, la précision des données, la qualité des extrants et l'usage de la mémoire. Lors de cette étude comparative, l'entreprise a déployée les différents logiciels sur le même environnement technique de manière à assurer la validité de leur étude comparative. De plus, les mêmes entrants ont été utilisés. La Figure 1.2 présente les résultats de cette étude.

Software	Total Compute Time*	Mean SNP Concordance	Total # SNPs	# High Quality SNPs	% High Quality Imputed
IMPUTE2	23 hours	99.98%	668,180	620,792	92.9%
BEAGLE	213 hours	98.43%	484,023	320,991	66.3%
IMPUTE2 with Pre-phasing	8 hours	99.92%	668,180	297,196	44.5%
BEAGLE with Pre-phasing	34 hours	98.05%	484,023	293,890	60.7%
Minimac	18 hours	96.25%	667,870	450,790	67.5%

Figure 1.3 Étude de comparaison des outils d'imputation [9]

En se basant sur les informations illustrées à la Figure 1.3, il est possible d'observer que les logiciels qui sont performants, c'est-à-dire qui offrent le meilleur « Total Compute Time » et qui sont les plus précis, c'est-à-dire qui ont le meilleur « Mean SNP Concordance », fournissent la meilleure couverture (c.-à-d. le meilleur « Total SNPs »). Ils peuvent aussi fournir la qualité désirée (de l'anglais « High Quality SNPs ») ainsi qu'une bonne qualité des extrants (de l'angalis « High Quality Imputed »). D'après cette étude, le logiciel IMPUTE2 offre la meilleure performance, c'est-à-dire l'extrant qui comporte la meilleure qualité et la meilleure couverture. Il est à noter que l'activité de préphasage (de l'anglais « pre-phasing »), qui est une étape importante visant à améliorer grandement la performance, à un impact significatif sur la qualité des données extraites. Le préphasage est le processus qui permet d'estimer statistiquement les haplotypes, de chaque individu, dans son échantillon GWAS. La prochaine figure, la Figure 1.4, présente la différence entre la méthode d'imputation traditionnelle et la méthode courante.

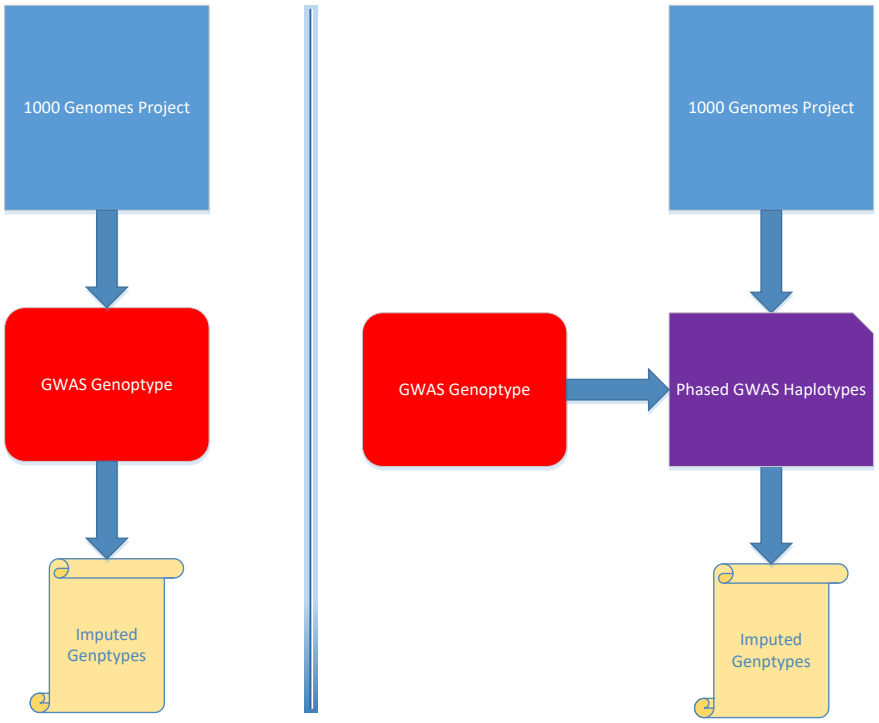


Figure 1.4 Imputation traditionnelle vs préphasage

1.4 Méthodologie d'imputation des outils

Cette section présente les différents outils ainsi qu'une vue d'ensemble de leur fonctionnement interne qui explique le fonctionnement d'une imputation génétique. En outre cette section permet d'identifier les algorithmes et la méthodologie utilisée par chaque logiciel. Les logiciels présentés dans cette section sont les plus utilisés. Les détails du fonctionnement de l'implémentation des logiciels (c.-à-d. le fonctionnement dans le code source) ne sont pas visés par cette étude synthèse. Il est important de noter que la documentation fournie varie entre ces logiciels. La plupart de la documentation est basée sur comment utiliser le logiciel. Afin d'accéder à la documentation et aux détails de l'implémentation, il est nécessaire de communiquer avec les auteurs/développeurs de ces outils. Les informations fournies dans cette section sont basées strictement sur les informations fournies par la documentation de ces outils et l'étude fournie par la section 1.3 de ce chapitre.

1.4.1 MACH

Le logiciel MACH est un outil d'imputation «basé sur la chaîne de Markov qui peut résoudre des haplotypes longs ou déduire des génotypes manquants dans des échantillons d'individus non apparentés. »[15]. MACH est un logiciel libre, sauf que le code source n'est pas disponible pour l'utilisation ou la consultation. Ce logiciel peut être déployé sur deux systèmes d'exploitation qui sont 1) Linux; et 2) Mac OS X. Pour l'inférence des génotypes, MACH utilise la méthodologie suivante:

- utilisation de l'algorithme de chaîne de Markov;
- l'algorithme associe, de façon aléatoire, une paire d'haplotypes à chaque individu qui est compatible avec les génotypes observés;
- les allèles sont attribués en fonction de leurs fréquences de la population pour les génotypes non typés;

- l’algorithme met à jour les configurations d'haplotypes en utilisant l'ensemble actuel des estimations d'haplotypes pour tous les individus en tant que modèles;
- il échantillonne les séries d'indicateurs en utilisant la chaîne de Markov;
- il répète la procédure de mise à jour un certain nombre de fois et compte combien de fois un génotype est échantillonné à une position particulière.

1.4.2 IMPUTE2

IMPUTE2 est un logiciel libre qui permet l'imputation des génotypes et le phasage des haplotypes. Ce logiciel est basé sur l’approche de Howie et coll. 2009 [16]. Les auteurs d'IMPUTE2 fournissent les fichiers binaires, ce qui permet d'exécuter l'outil. Pour le moment, cet outil peut être utilisé sur les systèmes d’opérations Linux et Mac OS X seulement. Selon la documentation, IMPUTE2 ne fournit pas d'interface utilisateur, donc les bioinformaticiens doivent programmer en ligne de commande afin d'exécuter une analyse. Les auteurs décrivent aussi deux utilitaires qui permettent: 1) de convertir des fichiers de type format d'appel variant (de l'anglais Variant Call Format VCF) en format de fichier génotype (.gen); ou 2) en un fichier de légende et un fichier haplotypes. Finalement, le logiciel IMPUTE2 utilise la méthode suivante pour l'inférence des génotypes:

- utilisation de l'algorithme basé sur le modèle de Markov caché (HMM);
- la séquence de paires d’haplotypes connus est traitée comme des états cachés;
- modélise la séquence de changement d'état caché le long de la séquence avec des taux de commutation en fonction de la carte de recombinaison estimée à partir des données de référence cachées;
- sur la base d’haplotypes connus, il prédit les génotypes non typés.

1.4.3 fastPHASE

fastPHASE est un logiciel libre basé sur le modèle de Scheet et Stephens 2006 [20] qui permet l'inférence des génotypes. Ce logiciel est seulement disponible en version exécutable,

ce qui ne permet pas la consultation du code source et les détails de son implémentation. Ce logiciel peut être déployé sur les systèmes d'exploitation Linux ou Mac OS X. Les auteurs du logiciel n'offrent pas d'utilitaires permettant la conversion des fichiers entrants ou des fichiers de sorties. Afin d'utiliser ce logiciel, il est important de consulter la documentation afin de mieux comprendre la structure des fichiers entrants. Pour l'inférence des génotypes, fastPHASE utilise la méthodologie suivante:

- utilise un algorithme de grappe d'haplotype (haplotype clustering algorithm);
- regroupe les haplotypes d'une région de population;
- modifie le long du chromosome sur la base d'un HMM;
- les génotypes manquants sont échantillonnés sur la base de la fréquence des allèles estimés à partir des haplotypes de référence;
- l'algorithme « Expectation-Maximization » est utilisé pour estimer les valeurs des paramètres; 6) en se basant sur les paramètres estimés, les génotypes manquants sont inférés.

1.4.4 BEAGLE

BEAGLE est un logiciel libre qui permet l'analyse à grande échelle «... avec des centaines de milliers de marqueurs génotypés sur des milliers d'échantillons »[18]. BEAGLE permet: 1) l'inférence des haplotypes pour des individus non apparentés; 2) l'inférence sporadique manquante; 3) d'inférer des marqueurs non génotypés qui ont été génotypés dans un groupe de référence; et 4) de détecter des régions génétiques qui sont homozygotes. Ce logiciel est développé en langage de programmation Java ce qui rend ce logiciel utilisable sur plusieurs plateformes. BEAGLE peut être modifié et redistribué sous la licence « General Public License (GNU) ». En outre, ce logiciel fournit plusieurs utilitaires qui permettent la préparation des fichiers entrants et traiter les fichiers extrants. En ce qui concerne l'imputation des génotypes, BEAGLE utilise la méthode suivante:

- utilise un algorithme de grappe d'haplotype (« haplotype clustering algorithm »);

- utilise une grappe d'haplotypes à chaque marqueur;
- définis un HMM pour trouver les paires d'haplotypes les plus probables en fonction des génotypes;
- le génotype le plus probable au locus non typé peut être généré à partir des paires d'haplotypes finales.

1.5 Discussion

Cette revue a permis d'identifier les outils d'imputation, de comparer leur performance, leur précision et la qualité des données extraite à la fin de ce processus. Le prochain tableau, le tableau 1.1, a pour but de fournir les caractéristiques de chaque logiciel présenté dans la section précédente de ce document.

Tableau 1.1.1 Comparaison des outils d'inférence (Imputation)

Fonctionnalité	MACH	IMPUTE2	fastPHASE	BEAGLE
Système d'exploitation	Linux, Mac OS X	Linux, Mac OS X	Linux, Mac OS X	Plateforme indépendante
Licence	Gratuit	Gratuit	Gratuit	Gratuit
Code Source	Non Disponible	Non Disponible	Non Disponible	Disponible
Interface utilisateur	Non Disponible	Non Disponible	Non Disponible	Non Disponible
Fichier entrant	Format personnalisé	Format personnalisé	Format personnalisé	Format personnalisé
Utilitaire de conversion	No	Oui	No	Oui
Performance	Moyen	Très Performant	Faible	Très Performant
Qualité d'Imputation (préphasage)	Bas	Moyen	Bas	Haut

Utilisation de mémoire vive	Moyen (7 GB)	Haut (10 GB)	Inconnue	Bas (2 GB)

Cette première étude comparative de logiciels d'imputation permettra de déterminer l'outil à utiliser lors d'une prochaine étape de cette recherche qui vise l'utilisation d'un de ces logiciels avec les formats génétiques et modèles de traitement pour l'informatique en nuage (c.-à-d. le projet ADAM de l'Université Berkeley) qui utilise des technologies BigData. Les éléments à considérer pour un choix judicieux sont les suivants: 1) l'accessibilité de l'outil; 2) sa documentation; 3) la qualité de l'imputation résultante; 4) sa performance; et 5) la présence d'utilitaires de conversion.

En ce qui concerne l'accessibilité, BEAGLE est l'un des outils le plus accessibles. Le code source peut être téléchargé et consulté. De plus, BEAGLE permet aux utilisateurs d'apporter les modifications au code source tout en respectant la licence GNU. Le code source des outils IMPUTE2, fastPHASE et MACH ne peut être consulté, ce qui ne permet pas l'analyse de la faisabilité durant les travaux subséquents. BEAGLE est implémenté en Java, ce qui permet par ailleurs de l'intégrer facilement avec des technologies BigData. Pour ce qui est de la documentation, BEAGLE fournit une meilleure documentation comparativement avec les autres logiciels. Les auteurs mettent à la disposition des développeurs une documentation qui permet d'identifier les types des fichiers utilisés, le contenu de ces fichiers, l'utilisation du logiciel et les commandes pertinentes. De plus, le code source de BEAGLE est bien documenté, ce qui permet de se familiariser rapidement avec son architecture interne et ses fonctionnalités. Concernant la qualité de l'imputation, BEAGLE propose une meilleure qualité comparée à IMPUTE2 avec une différence de 16.2% tel que présenté à la section 1.3 de ce chapitre. Concernant la performance, IMPUTE2 se distingue considérablement des autres solutions offertes. Mais le fait qu'IMPUTE2 n'est pas accessible, il ne peut être choisi pour la prochaine phase du projet de recherche.

Finalement, le dernier élément à considérer est la présence d'utilitaires de conversion efficaces. Ce critère est important, car le fait de fournir un utilitaire de conversion permettra

de transformer facilement le contenu des fichiers à traiter. Le fait que le logiciel BEAGLE est écrit en langage de programmation Java, il sera facile d'implémenter de nouveaux utilitaires, ou bien de les ajuster, afin de faciliter la lecture des données existantes. Un autre avantage notable, est le fait que le logiciel libre BEAGLE utilise les données du « 1000 Genomes Project » comme panneau de référence ce qui est aussi utilisé par le projet du Dr Hamet.

1.6 Conclusion

En conclusion, cette courte étude comparative permet de présenter une vue d'ensemble de l'imputation et de son objectif en ce qui concerne l'étude d'association génomique (GWAS). De plus, cette étude a permis d'identifier les critères de comparaison les plus pertinents à considérer en ce qui concerne la méthodologie d'inférence à utiliser dans le futur. Cette étude a aussi permis d'identifier que le logiciel libre BEAGLE est le logiciel à considérer pour les travaux futurs d'ajout d'un module d'imputation BigData au prototype actuel. Le prochain chapitre présente une approche proposée afin d'utiliser les technologies émergentes du BigData pour effectuer l'imputation génétique d'une manière plus efficace.

CHAPITRE 2

MODULE D'IMPUTATION Q(n)GENE

2.1 Objectifs

L'objectif de cette activité du projet de recherche appliquée est d'étudier une alternative au module d'imputation utilisé par l'équipe de recherche actuelle au laboratoire du Dr Hamet. Pour ce faire, les prochaines sections visent à décrire les logiciels actuels, ce qui permettra d'identifier les types de données existantes traitées par l'équipe de recherche (c.-à-d. fichiers entrants/extrants). La maitrise des formats de fichiers est une activité incontournable afin d'effectuer l'intégration future de ces données dans le modèle de données BigData du projet ADAM. Plus spécifiquement, cette intégration a été démarrée lors d'une première activité ADVANCE-to-ADAM effectuée, en automne 2016, par l'étudiant Diego Alvarez lors de son projet de maîtrise à l'ÉTS [27]. Les sections suivantes présentent une solution proposée en ce qui concerne le module d'imputation et les aspects à considérer durant la phase de la conception et de l'implémentation vers le format ADAM.

2.2 Technologies courantes

L'équipe de recherche du Dr Hamet utilise un ensemble d'outils (c'est à dire des logiciels bioinformatique libre) qui leur permet d'effectuer les tâches quotidiennes telles que, l'inférence statistique des génotypes (Imputation) et l'analyse des variations génétiques (GWAS). Les outils qui sont utilisés actuellement sont: 1) PLINK; 2) SNPTEST; et 3) IMPUTE2.

SNPTEST est « un programme pour l'analyse de l'association SNP unique dans des études génomiques. »[22]. L'étude de ce logiciel libre ne fait pas partie des objectifs de cette recherche et ne sera pas discuté dans ce document. Il est toujours possible de consulter les fonctionalités de ce logiciel à l'adresse internet: https://mathgen.stats.ox.ac.uk/genetics_software/snptest/snptest.html.

Une introduction au logiciel IMPUTE2 a été présentée au chapitre 1. Pour plus d'information, vous pouvez consulter les fonctionnalités de ce logiciel à l'adresse internet : http://mathgen.stats.ox.ac.uk/impute/impute_v2.html .

Pour ce qui est du logiciel libre PLINK, c'est un outil purement dédié à l'analyse des génotypes. PLINK a été développé par le Centre de recherche en génétique humaine (CHGR) avec le support de l'hôpital général du Massachusetts (MGH), le Broad Institute of Harvard et le Massachusetts Institute of Technology (MIT). Ce logiciel libre permet de: 1) gérer les données; 2) effectuer le contrôle de qualité; et 3) exécuter les tests d'associations. Ce logiciel ne fait pas partie des objectifs d'étude de cette recherche et ne sera pas discuté dans ce document. Les détails concernant les fonctionnalités de PLINK peut être consultées à l'adresse internet: https://www.cog-genomics.org/plink2.

2.3 Format des fichiers IMPUTE2

Actuellement, le logiciel libre IMPUTE2 infère les données à partir de la base de données Action contre le diabète et les maladies vasculaires (ADVANCE). Les fichiers qui sont générés par IMPUTE2 sont les fichiers de format « .gen » et « .sample ». Ces fichiers contiennent les données inférées lors d'une analyse. Ces fichiers sont ensuite analysés à l'aide des logiciel SNPTEST et PLINK. La prochaine figure, la figure 2.1, présente le contenu typique d'un fichier « .gen ».

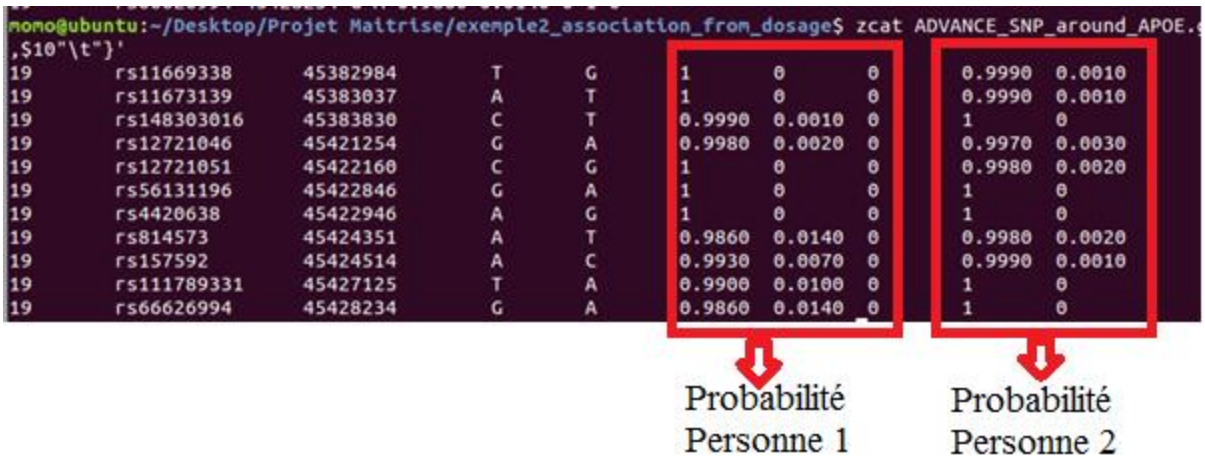


Figure 2.1 Format «.gen»

Les colonnes de la figure 2.1, représentent les données suivantes: 1) le chromosome (snp_id); 2) la variation (rs_id); 3) la position dans la chaîne d'ADN; 4) l'allèle A; et 5) l'allèle B. En ce qui concerne les trois colonnes représentant les probabilités, ils ont pour but de fournir la probabilité d'occurrence des trois génotypes, AA, AB et BB chez l'individu.

Le fichier au format « .sample » a pour but de fournir des informations sur les individus analysés. La prochaine figure, la figure 2.2, décrit un exemple du contenu de ce fichier.

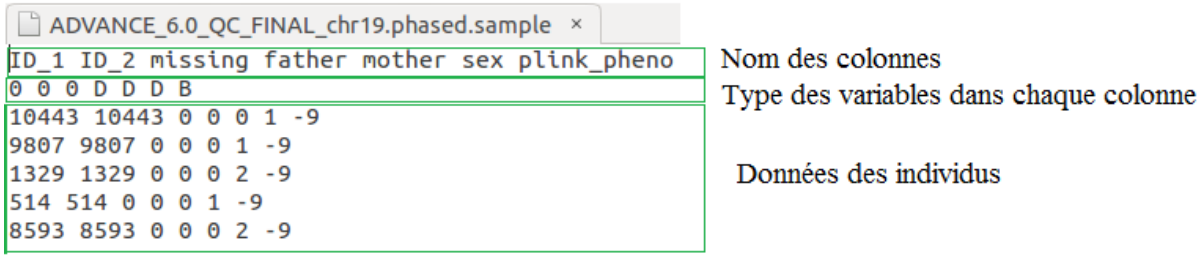


Figure 2.2 Format « .sample »

À la figure 2.2, les trois premières entrées doivent toujours être ID_1, ID_2 et manquantes. Elles indiquent le premier ID, le deuxième ID et la proportion de données manquantes pour chaque individu. Les entrées supplémentaires, sur cette ligne, doivent être les noms des covariables (ou phénotypes) qui sont inclus dans le fichier. L'exemple ci-dessus présente trois covariables nommées « father, mother et sex ». Il y a aussi la donnée concernant le phénotype, nommé plink_pheno.

2.4 Format VCF

Le format VCF est un format de fichier texte qui contient des marqueurs ainsi que des données concernant les génotypes. Le logiciel libre BEAGLE utilise ce format de fichier en entrant et en extrant lors d'une imputation génétique. La prochaine figure, la figure 2.3, pr.sente un exemple de fichier au format VCF.

#CHROM	POS	ID	REF	ALT	QUAL	FILTER	INFO	FORMAT	SAMP001	SAMP002
20	1291018	rs11449	G	A	.	PASS	.	GT	0/0	0/1
20	2300608	rs84825	C	T	.	PASS	.	GT:GP	0/1:.	0/1:0.03,0.97,0
20	2301308	rs84823	T	G	.	PASS	.	GT:PL	./.:.	1/1:10,5,0

Figure 2.3 Format VCF [25]

Les colonnes présentées à la figure 2.3, représentent les données suivantes: 1) le chromosome (CHROM); 2) la position dans la chaîne d'ADN (POS); 3) la variation (ID); 4)
l'allèle de référence exprimé sous la forme d'une séquence d'un ou plusieurs nucléotides par exemple "A" ou "AAC" (REF); 5) l'allèle alternatif exprimé en séquence d'un ou plusieurs nucléotides par exemple "A" ou "AAC" (ALT); 6) probabilité que l'allèle ALT est incorrectement spécifié (QUAL); 7) filtres de contrôle de qualité qui ont échoué ou passé (FILTRE); 8) des informations supplémentaires (INFORMATIONS); 9) liste des sous-champs par exemple "Génotype" ou "Génotype Likelyhood" (FORMAT).

2.5 Solution ADVANCE-to-ADAM avec IMPUTE2

Le projet de recherche ADVANCE-to-ADAM a expérimenté un processus qui permet de transformer les données existantes (données dans les bases de données ou générer par les outils d'imputation) en format Parquet-ADAM. L'importance de l'utilisation du format ADAM pour les études génétiques à l'aide d'outils BigData a été démontrée par l'équipe de recherche du laboratoire AMPLAB de l'Université Berkeley dès 2015. Les détails du schéma des données Parquet-ADAM est présenté en Annexe du rapport de recherche de l'étudiant Diego Alvarez. [27] Il existe plusieurs étapes de transformations dépendamment des types de données utilisées au départ. La prochaine figure, la figure 2.4, présente une vue d'ensemble de l'approche expérimentée lors de ce projet de recherche à l'aide du logiciel libre IMPUTE2.

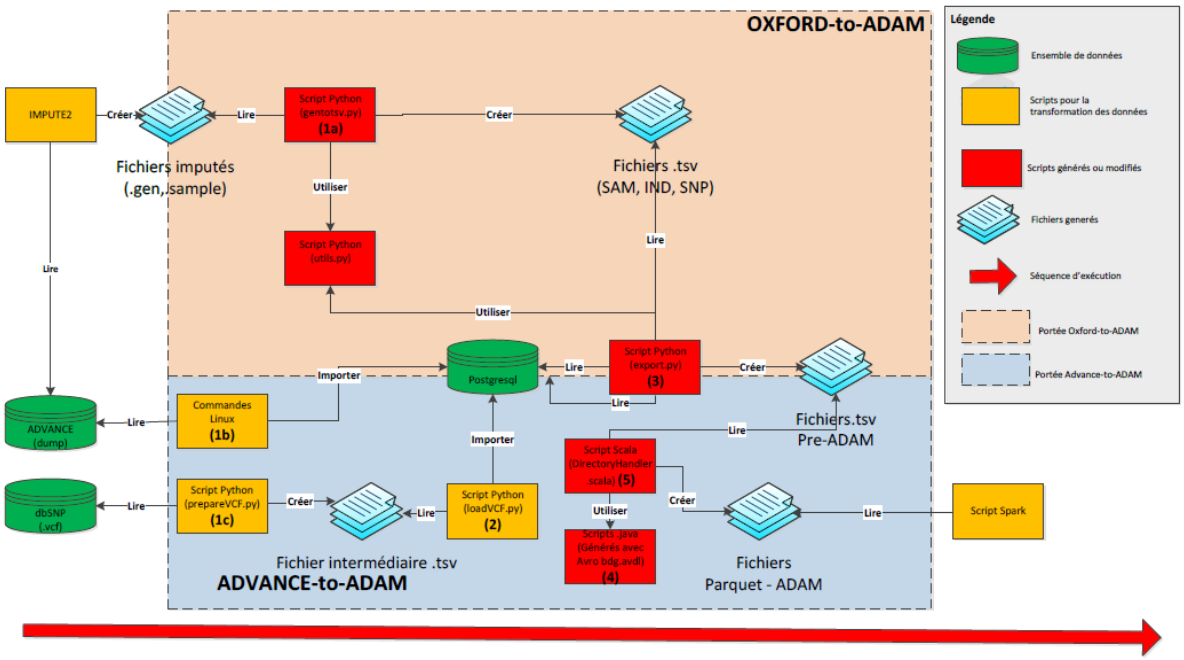


Figure 2.4 Solution ADVANCE-to-ADAM [27].

Présentement, IMPUTE2 lit les données à partir de la base de données ADVANCE. Ensuite, l'outil réalise une imputation avec les données lues de la base de données et génère des fichiers « .gen » et « .sample ». Les données générées par IMPUTE2 sont ensuite transformées afin d'être transformées en format Parquet-ADAM pour être lues avec Spark-ADAM.

2.6 Solution ADVANCE-to-ADAM avec BEAGLE

Cette section du rapport propose une solution alternative à considérer pour les travaux futurs et l'intégration avec ADAM. La solution suggéré propose d'utiliser le logiciel libre BEAGLE pour remplacer IMPUTE2, Cette proposition se base sur pour les critères suivants: 1) l'accessibilité de l'outil; 2) la qualité de l'imputation; 3) sa bonne documentation et 4) ses utilitaires de conversion.

Le fait que BEAGLE génère des fichiers de format VCF comme extrant, permet de réduire les travaux de transformations effectuées lors de l'expérience « OXFORD-to-ADAM ». Cela

minimiserais et même supprimerais les transformations intermédiaires en plus d'unifier les formats des fichiers. La figure 2.5 permet de voir la vue d'ensemble de la solution proposée.

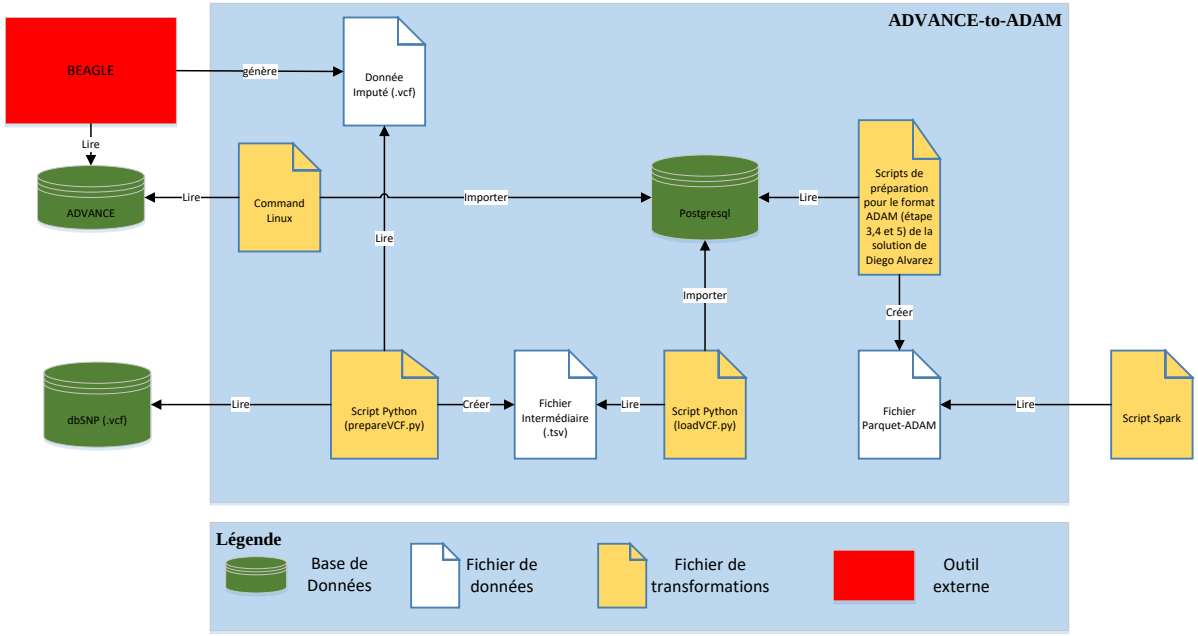


Figure 2.5 Solution ADVANCE-to-ADAM avec BEAGLE

L'avantage de cette solution est que les fichiers de transformations en langage de programmation Python et les fichiers intermédiaires (au format SAM, IND, SNP) sont aussi éliminés par cette nouvelle solution . Un autre avantage de l'utilisation de BEAGLE est qu'il sera possible, dans le futur, d'implémenter un module en Java qui effectuera la lecture des données à partir de la base de données ADVANCE. Une autre alternative serait de créer un script qui transforme les données ADVANCE en format VCF.

2.7 Conclusion

Ce chapitre a présenté les technologies actuelles et les types de fichiers à considérer dans les travaux subséquents. Il a aussi proposé une solution à envisager en ce qui concerne l'utilisation de BEAGLE avec le format ADAM. Cette activité de recherche a pris en considération la solution proposée par Diego Alvarez en ce qui concerne la transformation des données en Parquet-ADAM et les modifications à apporter si BEAGLE est utilisé.

CHAPITRE 3

OUTIL DE PROVENANCE

3.1 Problématique

Un des problème rencontré par les bioinformaticiens proviens de leur manque de connaissance en ce qui concerne les bonnes pratiques du génie logiciel. Leur compétence est plus axée sur la biologie plutôt que l'informatique. De plus, les bioinformaticiens ont typiquement peu de temps afin d'effectuer de l'architecture logicielle et de se former sur les nouvelles technologies et les pratiques de provenance modernes. Une autre problématique rencontrée lors de cette activité de recherche appliquée, est la capacité de reproduire les résultats d'une analyse génétique existante. Cela est principalement dû à la méthodologie utilisée lors d'une analyse qui ne permet pas facilement de la reproduire après plusieurs mois. La méthode essai-erreur actuellement utilisée ne leur permet pas facilement de bien documenter les étapes d'une analyse, car il existe beaucoup de variation dans les paramètres et les transformations lors de la recherche sans oublier les avancements technologiques rapides des logiciels qu'ils utilisent. Conséquemment, les bioinformaticiens n'ont pas une méthodologie qui leur permet d'identifier facilement les données sources ainsi que les transformations effectuées lors d'une analyse génétique. Un autre problème identifié est le manque de contrôle sur les fichiers intermédiaires et les scripts de transformation utilisés. Tous ces facteurs peuvent impacter la capacité et l'effort nécessaire afin de reproduire une analyse existante.

3.2 Objectifs de la solution

Une solution possible afin d'adresser les problématiques présentées à la section 3.1 est de capturer et sauvegarder la recette de transformation, c'est-à-dire conserver les détails de l'algorithme (c'est-à-dire du code source exécuté) durant une analyse. De plus, il faut déterminer les entrées et les sorties des transformations effectuées par les bioinformaticiens tout au long d'une analyse. La figure 3.1, présente une proposition de solution.

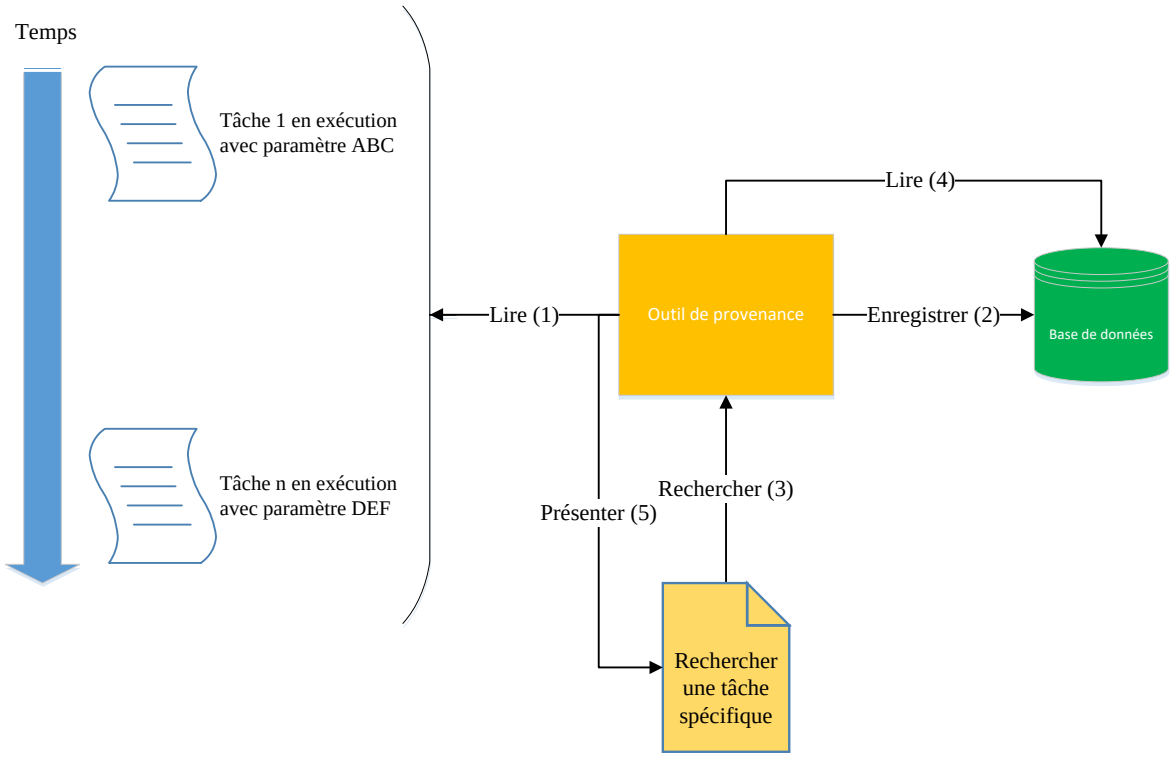


Figure 3.1 Solution outil de provenance

Afin de mieux comprendre la figure ci-dessus, voici une description simple des étapes d'exécution proposées par la solution d'outil de provenance:

1) L'outil de provenance (c.-à-d. le logiciel) lit les informations des tâches à travers le temps. Par exemple, si la tâche 1 est exécutée au temps1, l'outil sauvegarde ces données. De même, si la tâche1 est exécutée à temps ***n***, l'outil enregistre ces données sans écraser les données antérieures.
2) L'outil de provenance sauvegarde les informations dans une base de données pour être consulté plus tard.
3) L'usager demande la recherche d'une tâche.
4) L'outil lit les données enregistrées et recherche la tâche demandée.
5) L'outil de provenance retourne la tâche avec les détails afin d'être analysé par les bioinformaticiens.

3.3 Les défis de la solution

Afin de fournir une solution d'outil de provenance permettant de résoudre les problématiques identifi.es, il est important d'identifier les défis et les obstacles qui devront être surmontés lors de la conception de la solution proposée. Voici les défis identifiés:

1) Fournir une solution qui s'exécute en temps réel et qui permet d'enregistrer les tâches, les transformations, les entrants et les extrants de chaque tâche;
2) La solution ne doit par interférer loirs de l'exécution de la tâche effectuée. C'est-à-dire une méthode discrète (de l'anglais unobtrusive) doit être implantée;
3) Enregistrer les tâches et les transformations de manière à ce qu'il soit possible de faire la recherche en utilisant le nom de la tâche. C'est-à-dire rechercher toutes les transformations effectuées, pour une même tâche, même si les transformations n'ont pas été effectuées consécutivement;
4) Enregistrer chaque modification effectuée durant l'exécution d'une tâche. C'est-à-dire que si une modification est apportée à un paramètre, cette information doit être sauvegardée afin de pouvoir la consulter dans le futur;
5) La structure des éléments à sauvegarder, soit la manière dont les données sont conservées.

Le choix des technologies à utiliser pour implanter cette solution s'avère un autre défi de cette recherche. Le fait d'utiliser une technologie spécifique pourrait avoir un impact sur la manière dont l'information est enregistrée. De même, la technologie choisie pourrait être limitée à enregistrer les entrants et les extrants. C'est-à-dire omettre des détails des transformations exécutées. Ces décisions sont expliquées dans le chapitre suivant.

3.4 Conclusion

Ce chapitre a présenté la problématique en ce qui concerne la sauvegarde et la reproduction des analyses génétiques effectuées par les bioinformaticiens. Les pistes de solution présentées dans ce chapitre pourraient aux bioinformaticiens d'enregistrer et de consulter les tâches effectuées lors d'une analayse. De plus, cette proposition de solution permettrait la

consultation des détails de ces tâches telles que: 1) les paramètres; 2) les algorithmes/code source exécuter; 3) les entrants; ainsi que 4) les extrants. Le chapitre suivant présente l'environnement de travail ainsi que les technologies utilisées lors de cette recherche pour le développement d’une preuve de concept d’un prototype de provenance pour les analyses génétiques.

CHAPITRE 4

ENVIRONNEMENT DE TRAVAIL

4.1 Objectifs

L'objectif de ce chapitre est de présenter les outils et l'environnement de travail choisis pour la conception et l'implémentation d'une simple preuve de concept de solution potentielle de provenance pour les analyses génétiques. En outre, ce chapitre précise l'objectif des outils et le but de leur utilisation. Le système d'exploitation utilisé pour ce prototype est Linux Ubuntu 15.10. Le choix de cet environnement est dû au fait que les technologies BigData opèrent bien dans l'environnement Linux.

4.2 ZeppelinExtractor

Le projet ZeppelinExtractor est un logiciel libre de provenance qui permet l'extraction des données de l'interpréteur Zeppelin [29]. De plus, ce logiciel libre permet de sauvegarder les données dans une base de données PostgreSQL. Le langage de programmation utilisé pour le développement du prototype logiciel est le langage Java. Les détails de l'implantation du prototype logiciel seront discutés dans le prochain chapitre, le chapitre 5.

4.3 Zeppelin

Zeppelin un est un cadriciel (de l'anglais framework) gratuit qui permet: 1) l'ingestion des données (c'est-à-dire importer/exporter des données afin de les utiliser immédiatement ou de les sauvegarder dans une base de données.); 2) la découverte des données; 3) l'analyse des données; et 4) la visualisation des données. L'intérêt de l'utilisation de Zeppelin est dans son intégration actuelle avec Apache Spark, ce qui permet l'injection automatique de SparkContext et SparkSQL. De plus, cette intégration permet l'arrêt ou l'affichage d'un travail facilement. Les notions importantes du logiciel Zeppelin sont les suivantes: 1) le carnet (de l'anglais notebook) Zeppelin; et 2) le paragraphe (de l'anglais paragraphe) Zeppelin.

Chaque carnet est composé de 1 à n paragraphes. Un carnet peut être considéré comme un conteneur de paragraphe. Un paragraphe se compose de deux sections: 1) la section de code où le code source est inséré et 2) la section résultat où le résultat de l'exécution du code est présentée. L'annexe II présente un exemple de carnet simple exécutant « hellowWorld » avec un exemple de paragraphe résultant.

4.4 Configuration de Zeppelin

Une fois l'installation de Zeppelin effectuée [29], il est important de configurer le fichier « conf/zeppelin-env.sh » et le fichier « conf/zeppelin-site.xml ». Le fichier « zeppelin-env.sh » a pour but de spécifier les variables d'environnement, telles que: 1) la variable « JAVA_HOME »; et 2) la variable « SPARK_HOME ». Pour ce qui est du fichier « zeppelin-site.xml », il faut seulement spécifier l'adresse hôte « zeppelin.server.addr » et le port « zeppelin.server.port ».

4.5 Apache HTTP Client

Le projet Apache HttpComponents est «...responsable de la création et de la maintenance d'un ensemble d'outils de composants Java de bas niveau axés sur les protocoles HTTP. »[30]. Les composants HTTP d'Apache sont utilisés afin de solliciter l'hôte où Zeppelin est en exécution afin d'obtenir une réponse utilisant les données d'entrées désirées. La version utilisée pour cette preuve de concept est la version 4.5.3. Une fois le fichier « httpcomponents-client-4.5.3 » téléchargé, il faut seulement ajouter les fichiers « .jar » au « Classpath » du projet « ZeppelinExtractor ».

4.6 Jackson

Jackson est une librairie Java efficace qui vise à sérialiser des objets Java vers JSON et vice versa. Pour utiliser Jackson, il suffit de télécharger le ficher « jackson-1.9.0.jar » à l'adresse http://www.java2s.com/Code/JarDownload/jackson-all/jackson-all-1.9.0.jar.zip. Une fois le

fichier téléchargé, il faut seulement l'ajouter au « Classpath » du projet « ZeppelinExtractor ».

4.7 PostgreSQL

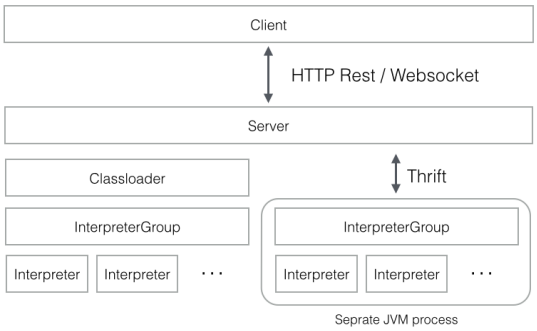
PostgreSQL est un système de base de données objet-relationnel libre. Ce système peut être déployé sur plusieurs systèmes d'exploitation, tels que Linux, Windows et Mac OS. PostgreSQL fournit des interfaces afin d'être intégré avec Java, Python et d'autres langages de programmation. La documentation de ce système est complète, ce qui permet une compréhension et une utilisation facile.

4.8 Apache Spark

Apache Spark est un système de grappe (de l'anglais cluster) rapide pour le traitement de l'information à grande échelle. Spark fournit des librairies (de l'anglais API) de haut niveau en Java, Scala, Python et R. L'avantage de Spark est dans le fait que ce cadriciel est «...100 fois plus vite en mémoire que Hadoop MapReduce et 10 fois plus rapide sur disque. »[32]. Un autre avatange est le fait que Spark peut être déployé d'une façon autonome, avec Hadoop ou en nuage (de l'anglais cloud).

4.9 Conclusion

Ce chapitre a présenté les technologies et les composants utilisés afin d'effectuer la preuve de concept. Le choix des technologies présenter dans ce chapitre est basé sur leur facilité d'intégration et la facilité de leur utilisation. Il était possible d'adopter une autre solution afin de fournir les fonctionnalités désirées. Cette solution consistait à s'accrocher à Zeppelin (de l'anglais hooking to Zeppelin). La solution consiste à créer un interprète Java et utiliser l'interface de Zeppelin afin de lire et extraire les données désirées. La figure suivante, la figure 4.1, permet de visualiser cette solution.



4.1 Architecture des interpréteurs Zeppelin [29].

L'interpréteur Zeppelin est un langage de programmation « backend ». « ... Par exemple, pour utiliser le code scala dans Zeppelin, vous avez besoin d'un interprète Scala. Tous les interprètes appartiennent à un groupe d'interprètes. Les interprètes du même groupe d'interprètes peuvent se référencer. Par exemple, SparkSqlInterpreter peut faire référence à SparkInterpreter pour en retirer SparkContext alors qu'il se trouve dans le même groupe.»[33]. De plus, l'interface « Interpreter » fournit les méthodes nécessaires qui permettent de démarrer et arrêter une tâche.

Le chapitre suivant présente l'implémentation et la conception d’un prototype « ZeppelinExtractor », ainsi que l'utilisation des technologies présentées.

CHAPITRE 5

CONCEPTION DE LA SOLUTION

5.1 Objectifs

L'objectif de ce chapitre est de fournir des détails en ce qui concerne la conception et l'implémentation d'un prototype logiciel fondé sur « ZeppelinExtractor ». En outre, ce chapitre présente: 1) l'architecture de haut niveau; et 2) les détails de la conception et de l'implémentation des technologies retenues au chapitre précédent.

5.2 Architecture

Cette section a comme but de présenter l'architecture de « ZeppelinExtractor », ainsi que l'interaction entre les différents modules de la solution proposée. La prochaine figure, la figure 5.1, permet de visualiser cette architecture.

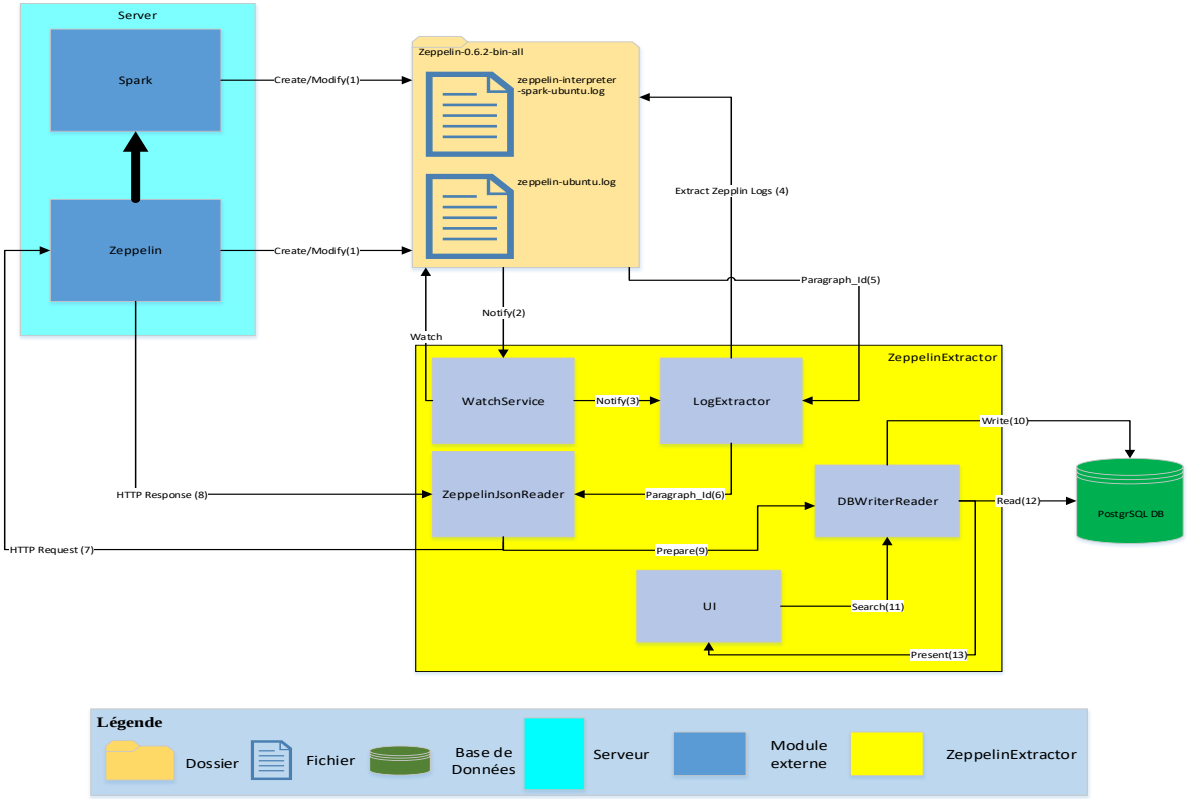


Figure 5.1 Architecture ZeppelinExtractor et l'interaction des différents modules

Pour capturer les activités et données utilisées lors d'une analyse génétique, un environnement de travail est proposé aux bioinformaticiens. Les modules Zeppelin et Spark sont deux cadriciels utilisés afin de capturer les tâches (et les données) exécutées par les bioinformaticiens lors d'une analyse génétique. Ces modules peuvent être déployés sur un serveur ou localement. Les bioinformaticiens créent ainsi des carnets et des paragraphes directement dans le module Zeppelin et ce dernier utilise le cadriciel Spark afin d'exécuter les tâches d'une analyse génétique dans un SparkContext.

Le répertoire « Zeppelin-0.6.2-bin-all » est un dossier dans lequel se trouve tous les fichiers de configurations Zeppelin, les scripts de démarrage et les fichiers journaux (de l'anglais « log files ») générés par Zeppelin et Spark. Les 2 fichiers les plus intéressants à considérer sont: 1) le fichier « zeppelin-ubuntu.log »; et 2) le fichier « zeppelin-interpreter-spark-ubuntu.log ». Le premier fichier, le fichier « zeppelin-ubuntu.log », est un journal créé par Zeppelin afin d'enregistrer les activités réalisées par un carnet. Ce fichier contient des informations telles que le temps de démarrage et d'arrêt du module Zeppelin, le temps d'exécution d'un paragraphe, l'identifiant unique d'un paragraphe et les tâches exécutées pour ce paragraphe. Ce fichier est créé et modifié par le module Zeppelin à chaque fois que le « deamon » Zeppelin est démarré ou arrêté. En outre, ce fichier est modifié lorsqu'un carnet est créé ou lorsqu'un paragraphe est exécuté. Le fichier «zeppelin-interpreter-spark-ubuntu.log » est généré par le module Spark et contient les informations en ce qui concerne les tâches exécutées par « SparkContext ». Ce fichier est créé et modifié lorsqu'un paragraphe ou une tâche est exécuté par le carnet afin de visualiser les extrants du paragraphe.

Le module ZeppelinExtraxtor est un module qui a pour but de surveiller les fichiers journaux et d'enregistrer les tâches exécutées par Zeppelin. De plus, ce composant extrait les informations d'un paragraphe exécuté via l'API REST de Zeppelin. Une autre tâche effectuée par ce module est la préparation des données afin d'être sauvegardées dans la base donnée.

En outre, ce module permet aux utilisateurs de rechercher les tâches sauvegardées et de consulter les informations connexes au paragraphe exécutées antérieurement.

5.3 Implémentation du module ZeppelinExtractor

Cette section du chapitre a pour but de fournir les détails de l'implémentation du module ZeppelinExtraxtor et de démontrer l'utilisation des outils et librairies externes.

La première tâche accomplie par le ZeppelinExtraxtor est une tâche qui s'exécute chaque seconde et effectue des requêtes « http » sur le serveur ou Zeppelin est en exécution afin de rechercher les carnets et leur paragraphe respectif. La tâche « NoteBookWatcher.java » permet d'extraire les informations des carnets existants ainsi que les carnets nouvellement créés par un usager. Les informations tirer de ces requêtes sont: 1) les identifiants des carnets (de l'anglais « notebook id »); 2) les noms des carnets (de l'anglais « notebook name »); et 3) les identifiants des paragraphes (de l'anglais « paragraph id ») existant dans chaque carnet. Le fait de cartographier (de l'anglais « to map ») les identifiants des paragraphes à leur carnet facilite la recherche lorsqu'un paragraphe est exécuté et la requête pour extraire l'information de l'exécution est envoyée au serveur. Il était possible d'extraire les informations des carnets et des paragraphes existants sans avoir à créer une tâche (de l'anglais « thread »). C'est à dire, faire une requête pour tirer les informations une fois l'application démarre seulement. Le désavantage de cette solution est que si jamais l'usager crée un nouveau carnet durant l'exécution de l'application, il était impossible de savoir qu'un nouveau carnet était ajouté. En utilisant une tâche qui s'exécute périodiquement ce problème était éliminé. La figure suivante, la figure 5.2, permet de présenter la méthodologie utilisée pour effectuer des requêtes « http ».

```
try {
    // URI to get the list of note books
    uri = new URIBuilder()
            .setScheme("http")
            .setHost(host)
            .setPath(path)
            .build();
} catch (URISyntaxException e) {
    e.printStackTrace();
}

httpget = new HttpGet(uri);
// execute the uri and get response
HttpResponse response = client.execute(httpget);
// use of Jackson to deserialize response
mapper = new ObjectMapper();
// read body to get the notbooks list
JsonNode myObjects = mapper.readTree(response.getEntity().getContent()).get("body");
```

5.2 Requête HTTP afin d'extraire les carnets existants

Afin de convertir les objets « Json » en objet Java, la librairie « Jackson » était utilisée. La figure 5.3 permet de démontrer l'utilisation de cette librairie. Il est important de mentionner que les données des carnets sont sauvegardées en permanence tout au long de la vie de l'application afin d'être disponibles au besoin. La classe qui permet de sauvegarder les données en permanence est la classe « DataUtils.java »

```
// JSON from URL to Java Object
NoteBook book = mapper.readValue(jsonNode, NoteBook.class);
```

5.3 Conversion des objets Json en objet Java

De plus, il existe deux tâches (de l'anglais « threads ») qui s'exécutent immédiatement au démarrage. La première tâche « SparkFileWatcher.java » qui a pour but de surveiller les mises à jour effectuées sur le fichier « zeppelin-interpreter-spark-ubuntu.log ». Le fichier « zeppelin-interpreter-spark-ubuntu.log » contient l'information en ce qui concerne les données entrantes (de l'anglais « input ») utilisées par Zeppelin. Tandis que la deuxième tâche « ZeppelinFileWatcher.java » a comme but de veiller sur le fichier « zeppelin-ubuntu.log ». Le fichier « zeppelin-ubuntu.log » contient l'identifiant du paragraphe exécuter par l'interpréteur Zeppelin. Des exemples de ces fichiers sont fournis en annexe. À chaque mise à jour

effectuée sur un de ces fichiers, les tâches observatrices utilisent les informations pertinentes et nécessaires pour l'extraction des données des tâches exécutées par l'interpréteur Zeppelin. Afin d'extraire les données pertinentes, à partir des fichiers journaux, chaque ligne est examinée pour savoir si elle renferme des mots-clés. À partir des mots-clés, l'application peut déterminer si la ligne contient les données nécessaires ou non. La prochaine figure, la figure 5.4, permet de visualiser les mots-clés recherchés.

```
if (line != null && line.contains("run paragraph")) {
    new ZeppelinExtractor(line);
}
```

5.4 Exemple des mots-clés rechercher dans un texte

À l'instant où le mot-clé est repéré dans la ligne ajoutée au fichier de journal (de l'anglais log file), le texte est ensuite envoyé aux classes d'extraction qui permettent d'extraire les données utilisées lors de la requête HTTP envoyée au serveur. Dans le module ZeppelinExtraxtor il existe deux classes d'extractions: 1) la classe « ZeppelinExtractor.java »; et 2) la classe « SparkExtractor.java ». Afin d'extraire seulement les données pertinentes, il a été nécessaire d'utiliser la fonction d'expressions régulières de Java. Cette fonction permet d'extraire les données désirées à partir d'un texte ou une chaîne de caractère. La prochaine figure, la figure 5.5, permet de visualiser comment ces données sont extraites.

```
/**
 * Method to extract the paragraph id using regex from the log
 * @param log
 */
private void extractLog(String log){

    Pattern pattern = Pattern.compile(".*-\\s+run\\s+paragraph\\s+([0-9-_]*).*");
    Matcher matcher = pattern.matcher(log);

    if(matcher.find()){
        String paragraphId = matcher.group(1);
        DataUtils.setParagraphId(paragraphId);
        DataUtils.setZeppelinFileRead(true);
    }
```

5.5 Extraction des données utilisées lors de la requête HTTP

Une fois que les mises à jour des fichiers journaux sont effectuées et que les données nécessaires pour les requêtes sont extraites, l'application effectue une requête sur le serveur pour retirer les informations afin d'être sauvegardées dans la base de données. La prochaine figure présente un fragment du code source démontrant cette fonctionnalité.

```
try {
    //query the data for the paragraph that was ran
    paragraphContent = ZeppelinJsonReader.getParagraphContent(DataUtils.getHost(),
            "/api/notebook/"+noteBookId+"/paragraph/"+paragraphId);
} catch (UnsupportedOperationException | IOException e) {
    e.printStackTrace();
}

//gets the executed code
JsonNode code = paragraphContent.get("text");
//gets the result of the executed code
JsonNode result = paragraphContent.get("result");
JsonNode output = result.get("msg");

//gets the date of when the code/result was created
JsonNode dateCreated = paragraphContent.get("dateCreated");
//gets the date of when the code/result was updated
JsonNode dateUpdated = paragraphContent.get("dateUpdated");
//gets the date of when the execution started
JsonNode dateStarted = paragraphContent.get("dateStarted");
```

5.6 Fragment du code source démontrant les informations retirées lors de l'exécution d'un paragraphe

Lorsque les informations du paragraphe exécuté sont retirées, une connexion à la base de données est initiée afin d'insérer et sauvegarder ces données. La prochaine figure, la figure 5.7, fournit un autre fragment du code source démontrant l'insertion des données dans la base de données.

```
//Connection to postgreSql Database
try {
    Class.forName("org.postgresql.Driver");
    conn = DriverManager.getConnection("jdbc:postgresql://127.0.0.1:5432/zeppelindb", "zeppelin", "zeppelin");
    conn.setAutoCommit(false);
     System.out.println("Opened database successfully");
     Statement stmt = conn.createStatement();

     //clean data
     //TODO: depending on the data , some of the output/input data might need
     //cleanup before inserting in the DB, that is due to special characters like
     // single quotes(') or double quotes ("")
     String codeDb = code.getTextValue().replaceAll("'", " ");
     String outputDb = output.getTextValue();

     String sql = "INSERT INTO DATA (notebookid,notebookname,paragraphid,code,output,input,datecreated,dateupdated,datestarted) "
        + "VALUES ('" + noteBookId + "','" +
                    noteBookName + "','" +
                    paragraphId + "','" +
                    codeDb + "','" +
                    outputDb + "','" +
                    inputFile + "','" +
                    dateCreated + "','" +
                    dateUpdated + "','" +
                    dateStarted + "');";
    stmt.executeUpdate(sql);
    stmt.close();
    conn.commit();
    System.out.println("Query executed successfully");
    conn.close();
```

5.7 Insertion de données dans la base de données

Il est à noter que certaines des données reçues doivent être affinées afin d'enlever les caractères spéciaux, tels que la citation simple « ' », car cela peut engendrer une exception lors de l'insertion dans la base de données.

5.4 Les données Zeppelin

L'objectif de cette section du chapitre est de présenter les données manipulées durant le processus de l'extraction. Tel que mentionné, les données sont obtenues à partir des fichiers de journaux ainsi qu'à partir des réponses « http » reçues de l'interpréteur Zeppelin. Cette section présente les requêtes qui peuvent être effectuées dans les navigateurs web afin de faciliter la tâche des développeurs.

Afin de déterminer la liste des carnets existants ainsi que leur identifiant (nom et identifiant alphanumérique) à partir du navigateur web, il suffit d'insérer l'URL suivant:

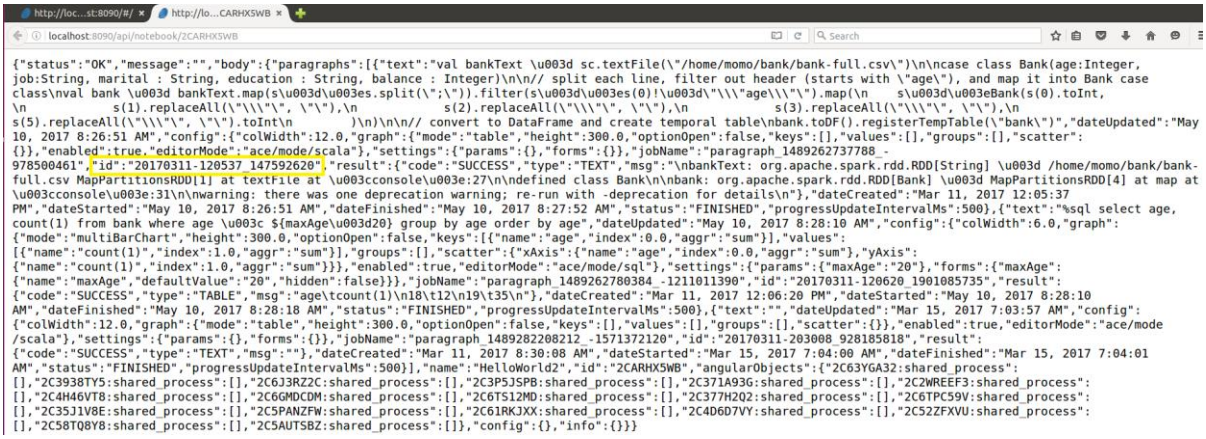
1) http://localhost:8090/api/notebook

En insérant cette requête le navigateur retourne la réponse suivante:

1) {"status":"OK","message":"","body":[{"id":"2CARHX5WB","name":"HelloWorld2"},{"id":"2BWJFTXKJ","name":"R Tutorial"},{"id":"2A94M5J1Z","name":"Zeppelin Tutorial"},{"id":"2BQA35CJZ","name":"Zeppelin Tutorial: Python - matplotlib basic"},{"id":"2C6PKDRD3","name":"helloWorld"},{"id":"2CFU6NXU3","name":"helloworld3"}]}

Tel que présenté à la section 5.3, les informations des carnets sont sauvegardé afin d'être utilisées par les requêtes HTTP.

Pour déterminer les paragraphes qui existent dans chaque carnet, la requête suivante peut être effectuée: http://localhost:8090/api/notebook/2CARHX5WB. Dans cette requête, l'identifiant numérique du carnet est nécessaire afin d'extraire les informations concernant les identifiants des paragraphes. La figure suivante, la figure 5.8, démontre la réponse retournée par le navigateur. La partie surlignée en jaune représente un identifiant d'un paragraphe.

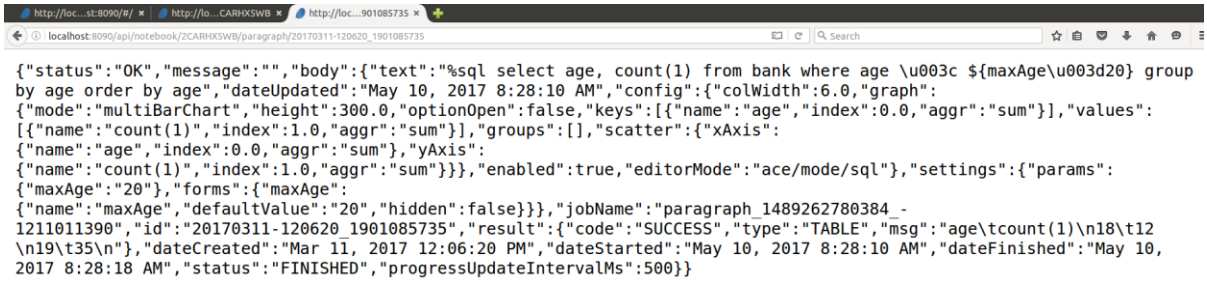


5.8 Requête qui permet d'identifier les paragraphes existants dans un carnet

La prochaine requête a pour but de rechercher les informations du paragraphe exécuté. Cette requête permet d'extraire les données telles que: 1) le code exécuté; 2) le résultat de l'exécution du code; 3) la date de création du paragraphe; 4) la date de modification du paragraphe; 5) et la date de la dernière modification effectuée sur le paragraphe. L'exemple suivant permet de visualiser cette requête:

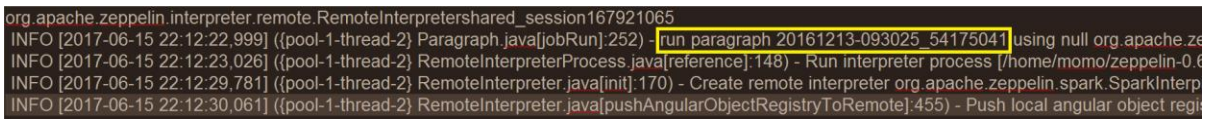
1) http://localhost:8090/api/notebook/2CARHX5WB/paragraph/20170311-120620_1901085735

L'identifiant du carnet ainsi que l'identifiant du paragraphe sont présents dans la requête ci-dessus. La figure suivante, la figure 5.9, présente la réponse reçue en exécutant cette requête.



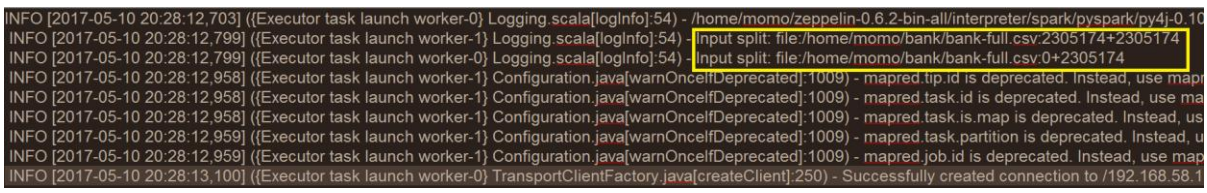
5.9 Requête qui permet d'extraire les informations d'un paragraphe exécuter

En ce qui concerne les informations retrouvées dans les fichiers journaux, il existe deux types de données. La première permet d'identifier le paragraphe exécuté. Afin d'extraire cette information, il suffit de consulter le fichier « zeppelin-ubuntu.log ». La prochaine figure, la figure 5.10, démontre cette information.



5.10 Exemple de fichier journal contenant l'identifiant du paragraphe exécuter

Le deuxième type de données recherchées dans les fichiers journaux, est la donnée permettant d'identifier le fichier entrant. Cette information peut être consultée dans le fichier « zeppelin-interpreter-spark-ubuntu.log ». La figure 5.11 présente un exemple de ce fichier ainsi que l'information rechercher.



5.11 Exemple de fichier journal contenant le fichier entrant

5.5 Base de données

Cette section du rapport a pour but de fournir des détails en ce qui concerne la création de la base de données, la création de la table qui permet de sauvegarder les données ainsi que la présentation des colonnes de cette table.

Afin de créer la base de données à utiliser avec l'application ZepplinExtractor, il faut s'assurer que PostgreSQL est installé. Une fois l'installation effectuée, il suffit d'exécuter les commandes suivantes afin de créer la base de données et la table:

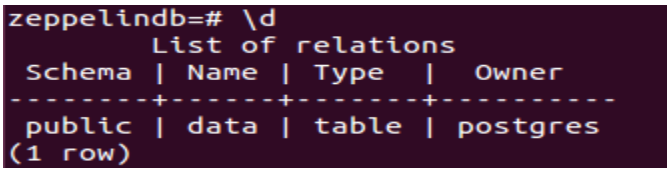
1. sudo -su postgres
2. psql -c "CREATE USER zeppelin WITH PASSWORD 'zeppelin';"
3. psql -c "CREATE DATABASE zeppelindb OWNER zeppelin;"

Les instructions fournies ci-dessus permettent de créer la base de données ainsi que de spécifier l'usager et le mot de passe.

Une fois la BD créée, la table « data » peut être créée avec les colonnes désirées. Voici la requête SQL qui permet de créer cette table:

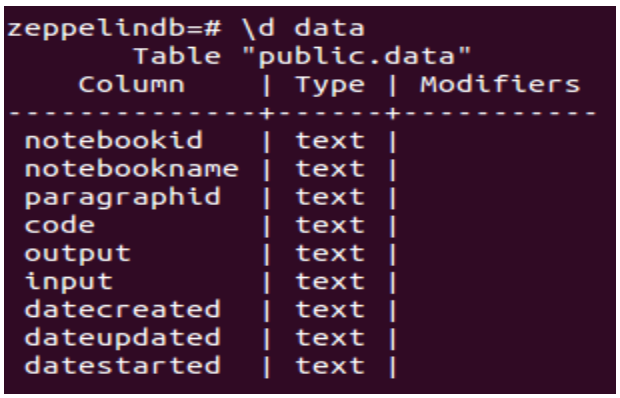
1. CREATE TABLE data (notebookid text, notebookname text, paragraphid text, code text, output text, input text, datecreated text, dateupdated text, datestarted text);

La figure suivante fournit un exemple qui permet de valider la création de la table « data ».



5.12 Validation de la création de la table « data »

La prochaine figure, la figure 5.13, fournit un exemple permettant la validation des colonnes dans la table.



5.13 Validation des colonnes dans la table « data »

Le fait de ne pas spécifier des clés primaires dans la table n'était pas nécessaire dans la solution actuelle, car il existe une seule table. De plus, pour permettre l'insertion de données similaires, il était nécessaire de négliger les mots-clés tels que le mot « UNIQUE ». Cela est dû au fait qu'un même paragraphe peut être exécuté en consécution et la seule différence dans ce cas est le temps d'exécution. Cette solution permet de fournir une meilleure flexibilité en ce qui concerne l'insertion de données. Puisque les données extraites sont toutes des chaînes de caractère, il était plus facile d'utiliser le type les colonnes « TEXT ». Cela est dû au fait que la longueur des chaînes de caractères extraites est imprévisible, et peut varier d'un paragraphe à un autre. La prochaine figure, la figure 5.14, fournit un exemple d'une table avec des données réelles.

Tableau 5.1 Exemple de données dans la table « data »

notebookid	**notebookname**	**paragraphid**	**code**	**output**	**input**	**datecreated**	**dateupdated**	**datestarted**
2CFU6NXU3	helloworld3	20170509-204026_128596010	%sql select age, count(1)	age count(1) 18 12 19 35	file:/home/momo/ban	May 9, 2017 8:40:2	May 9, 2017 9:13:5	May 9, 2017 9:13:

			from bank where age < ${maxAge=20} group by age order by age		k/bank-full.csv	6 PM	0 PM	50 PM
2CARHX5WB	HelloWorld2	20170311-120620_1901085735	%sql select age, count(1) from bank where age < ${maxAge=30} group by age order by age	age count(1) 18 12 19 35 20 50 21 79 22 129 23 202 24 302	file:/home/momo/bank/bank-full.csv	Dec 13, 2016 9:31:21 AM	May 9, 2017 9:14:06 PM	9, 2017 9:14:06 PM

5.6 Conclusion

Ce chapitre a présenté l'architecture, les détails de la conception et de l'implémentation, incluant l'utilisation des librairies externes, en ce qui concerne la proposition d’une solution de provenance pour les analyses génétiques fondées sur ZepplinExtractor. Cette section du rapport présente des fragments de code qui permettent de voir le prototype de solution implémenté, ainsi que la proposition de la manipulation des données d’analyses. Les détails de la conception de la base de données de la solution proposée ont aussi été présentés.

Le chapitre suivant présente les tâches réalisées durant ce projet de recherche appliquée, les difficultés rencontrées, les leçons apprises, ainsi que les recommandations pour des travaux futurs.

CHAPITRE 6

SYSTHÈSES DU PROJET

6.1 Objectifs

L'objectif de ce chapitre est de présenter le bilan de ce projet. De plus, ce chapitre permet de présenter les difficultés rencontrées durant le projet, les leçons apprises, ainsi que les recommandations pour des travaux subséquents.

6.2 Difficultés rencontrées

Les difficultés rencontrées durant ce projet sont majoritairement dues à la compréhension d'un nouveau domaine. Le domaine bioinformatique est un domaine complexe et demande beaucoup de connaissance en ce qui concerne la biologie et la médecine. La première partie de ce projet était la phase la plus ardue car il m'a fallu me familiariser avec ce domaine. Afin de mieux comprendre le domaine et la portée du projet de recherche, je devais faire mes propres recherches et lectures. Les articles et les recherches publiées en ligne sont souvent réalisées pour les experts du domaine qui détiennent déjà les connaissances nécessaires pour la compréhension du contenu. Pour surmonter cet obstacle, je devais être encadré par un expert du domaine afin de comprendre les termes, la méthodologie utilisée durant leur tâche et les outils utilisés présentement.

La deuxième difficulté rencontrée durant ce projet, était la compréhension des détails des outils utilisés par les bio-informaticiens et qui devais être améliorés. C'est-à-dire, les bio-informaticiens avaient la connaissance des outils. Il m'a donc fallu aussi faire mes propres recherches afin de mieux comprendre ces outils, les données utilisées et leur fonctionnement. De plus, les bio-informaticiens n'avaient pas les connaissances en ce qui concerne les bonnes pratiques du génie logiciel. Cela avait un impact négatif sur la maintenabilité de la solution, la performance ainsi que l'environnement d'opération où les solutions seront déployées.

En outre, les bio-informaticiens ne considérer pas la faisabilité d'une solution durant les discussions. C'est à dire que les membres de l'équipe du CRCHUM ont présenté leurs exigences sans prendre en considération la connaissance du domaine chez les développeurs, ni la disponibilité des données, ni la technologie disponible pour entamer une solution.

6.3 Leçons apprises

Le projet présenté dans ce rapport m'a permis de confirmer qu'il est primordial de comprendre le domaine d'affaires avant d'entamer une solution. Le fait de bien comprendre un domaine permet de: 1) confirmer la solution présentée; et 2) de fournir une solution qui répond aux exigences des usagers.

Une autre leçon retenue de cette recherche est que les bonnes pratiques de génie logiciel sont négligées en bio-informatique. Donc, il était nécessaire de ma part de promouvoir ces pratiques en fournissant de la documentation qui permet: 1) d'expliquer les décisions d'architecture; 2) de documenter le code source; et 3) d'expliquer la conception logicielle et les décisions prises pour une meilleure maintenance future.

En ce qui concerne l'aspect professionnel, ce projet m'a permis de travailler et d'expérimenter avec les technologies BigData et les nouvelles technologies qui peuvent être appliquées à plusieurs domaines, dont celui de la bioinformatique.

Enfin ce projet m'a permis d'améliorer l'aspect de gestion sur le plan personnel. Avec mes responsabilités vis-à-vis ma vie personnelle et professionnelle, je devais être très organisé et discipliné afin de respecter mes délais.

6.4 Recommandations

La section qui suit a pour but de présenter les fonctionnalités et les améliorations qui pourraient être apportées au projet ZepplinExtractor lors de projets subséquents.

Afin de fournir un logiciel selon les règles de l'art, il est important de créer de la documentation nécessaire, telle que : le document de vision et le document SRS (« Software Requirement Specification Document »). Ces documents permettraient de décrire mieux les caractéristiques et la portée du projet futur. De plus, ces artéfacts peuvent être utilisés comme référence et peuvent être consultés pour la maintenance ou l'évolution du logiciel.

En ce qui concerne les fonctionnalités, plusieurs améliorations peuvent être effectuées afin de concevoir un outil plus convivial. Pour l'instant, le prototype effectue les fonctionnalités de base comme: 1) l'observation des mises à jour effectuées sur les fichiers journaux existants; 2) l'extraction des identifiants des carnets; 3) l'extraction des identifiants des paragraphes; et 4) la sauvegarde des données dans la base de données.

Dans un nouvel environnement, les fichiers journaux n’existerons pas. C'est-à-dire que le fichier « zeppelin-interpreter-spark-ubuntu.log » et le fichier « zeppelin -ubuntu.log » ne seront pas disponibles dans le répertoire Zeppelin avant que l'usager démarre un paragraphe dans le carnet. L'amélioration peut être effectué en ajoutant une fonctionnalité qui permet de surveiller les fichiers journaux lors de leur création et leur modification. Afin d'implémenter cette utilité, une solution du « WatchService » qui est une classe de Java est fournie en annexe v.

Une autre amélioration importante a effectuer serait de fournir une interface utilisateur qui permet la recherche des tâches sauvegardées dans la base de données. La recherche peut être effectuée sur: 1) l'identifiant d'un carnet; 2) l'identifiant d'un paragraphe; 3) par date; 4) par un sous-chaine de caractère du code éxecuté; et, finalement 5) sur le nom du fichier d'intrant.

CONCLUSION

Ce projet consistait en l'exploration des outils d'imputation génétique, puis la comparaison de ces outils. Cette phase du projet a eu lieu au sein de l'équipe de recherche du docteur Pavel Hamet au Centre Hospitalier de l'Université de Montréal (CRCHUM). La deuxième partie de ce projet, consistait à l'analyse d'un logiciel libre, qui utilise la technologie BigData, qui permet au bio-informaticien de facilement répéter une expérience. De plus, ce projet consistait à implémenter un prototype de l'outil qui permet d'enregistrer les tâches des usagers ainsi que les données et les paramètres utilisés durant l'exécution.

Sur le plan professionnel, ces deux projets de recherche appliqués ont été une bonne expérience car ils m'ont permis d'apprendre un nouveau domaine. En outre, ces projets ont été avantageux sur le plan technique, car ils m'ont permis de m'initier à des nouvelles technologies telles que le BigData et l'interpréteur Zeppelin. Enfin, ces projets mon permis d'améliorer et d'acquérir mes capacités en ce qui concerne l'autonomie et la gestion du temps.

ANNEXE I

ADAM-PLINK GWAS DATA MAPPING

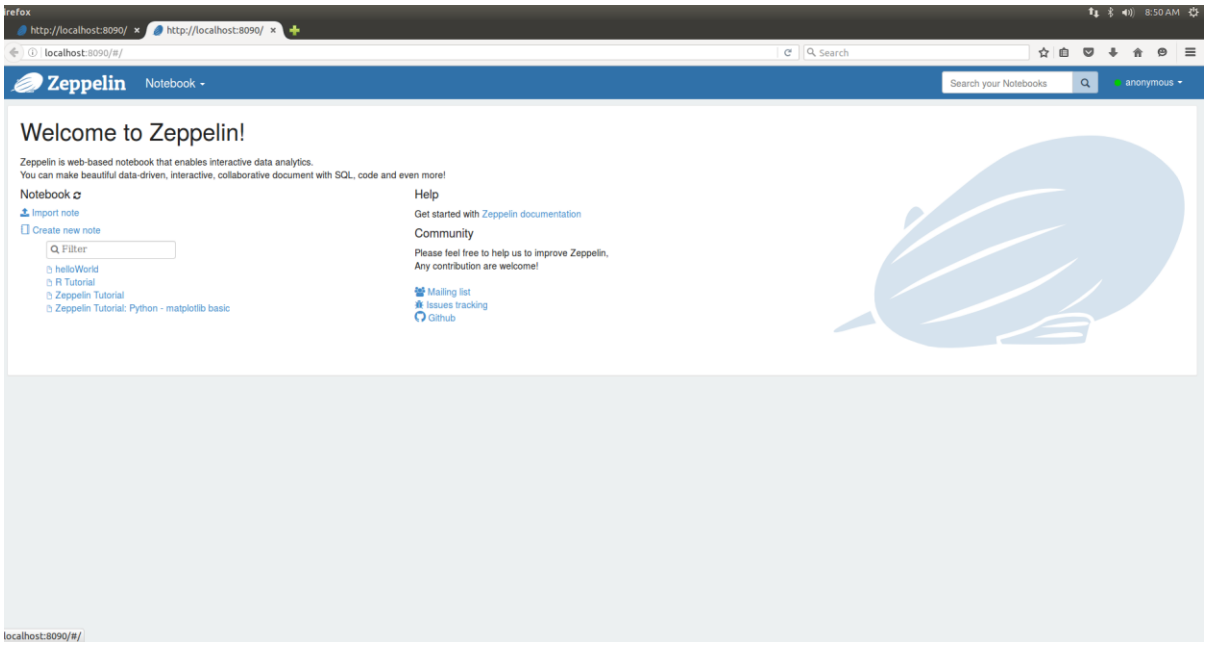
Plink Field	Data Type	Plink Description	.ASSOC	.FISHER	.model	.model .best .perm / mperm	.cmh	.cmh2	.homog	.T2	.T2 .perm	.T2 .mperm	.qassoc	.qassoc .means	.qassoc .gxe	.assoc .linear	.assoc .logistic	.assoc .set .mperm	.adjust
CHR	Int	Chromosome	Yes	Yes	Yes	Yes	Yes	Yes	Yes				Yes	Yes	Yes	Yes	Yes		Yes
SNP	Alphanumeric	SNP Identifier	Yes	Yes	Yes	Yes	Yes	Yes	Yes				Yes	Yes	Yes	Yes	Yes		Yes
BP	Binary	Physical position (base-pair)	Yes	Yes			Yes						Yes			Yes	Yes		
A1	Char	Minor allele name (based on whole sample)	Yes	Yes												Yes	Yes		
F_A		Frequency of this allele in cases	Yes	Yes					Yes										
F_U		Frequency of this allele in controls	Yes	Yes					Yes										
A2		Major allele name	Yes	Yes															
CHISQ		Basic allelic test chi-square (1df)	Yes		Yes	Yes	Yes		Yes										
P	Float	Asymptotic p-value for this test	Yes		Yes	Yes	Yes	Yes	Yes							Yes	Yes		
P		Exact p-value for this test		Yes															
OR	Float	Estimated odds ratio (for A1, i.e. A2 is reference)	Yes	Yes			Yes		Yes							Yes	Yes		
L95		Lower bound of 95% confidence interval for odds ratio	Maybe				Yes												
U95		Upper bound of 95% confidence interval for odds ratio	Maybe				Yes												
TEST	Alphanumeric	Type of test			Yes	Yes			Yes							Yes	Yes		
AFF		Genotypes/alleles in cases			Yes	Yes													
UNAFF		Genotypes/alleles in controls			Yes	Yes													
DF		Degrees of freedom for test			Yes	Yes			Yes										
A1		Minor allele code					Yes		Yes										
A2		Minor allele code					Yes		Yes										
CHISQ_CMH2		Cochran-Mantel-Haenszel test for IxJxK tables						Yes											
CHISQ_BD		Breslow-Day test						Yes											
P_BD		Asymptotic p-value						Yes											
N_A		Case allele count							Yes										

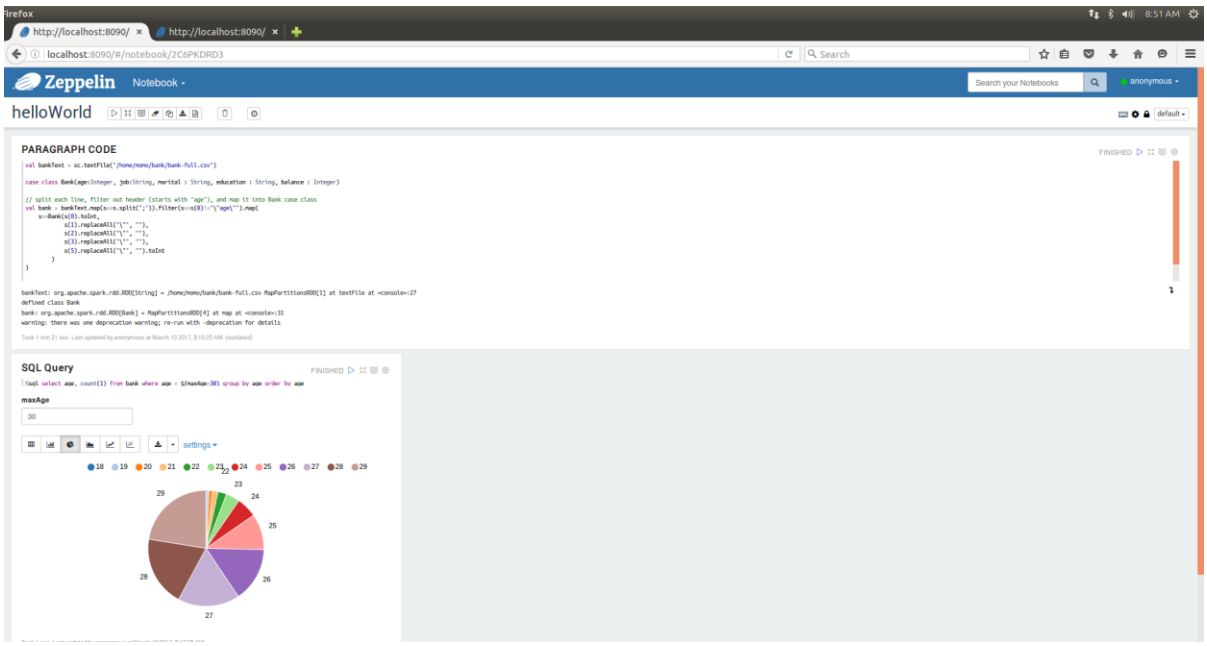
N_U		Control allele count							Yes										
SET		Set Name								Yes	Yes	Yes						Yes	
SIZE		Number of SNPs in this set								Yes	Yes	Yes						Yes	
F		F-statistic from Hotelling's test								Yes									
DF1		Degrees of freedom 1								Yes									
DF2		Degrees of freedom 2								Yes									
P_HOTEL		Asymptotic p-value								Yes									
EMP1	Float	Empirical p-value									Yes	Yes						Yes	
EMP2	Float	max(T) empirical p-value										Yes							
NR		Number of permutation replicates									Yes								
NMISS	Int	Number of non-missing genotypes											Yes			Yes	Yes		
BETA		Regression coefficient											Yes			Yes	Yes		
SE		Standard error											Yes						
R2		Regression r-squared											Yes						
T		Wald test (based on t-distribtion)											Yes						
P		Wald test asymptotic p-value											Yes						
Value		Description of next three fields												Yes					
G11	Float	Value for first genotype												Yes					
G12	Float	Value for second genotype												Yes					
G13	Float	Value for third genotype												Yes					
NMISS1		Number of non-missing genotypes in first group (1)													Yes				
NMISS2		Number of non-missing genotypes in second group													Yes				
BETA1		Regression coefficient in first group													Yes				
BETA2		Regression coefficient in second group													Yes				
SE1		Standard error of coefficient in first group													Yes				
SE2		Standard error of coefficient in second group													Yes				
Z_GXE		Z score, test for interaction													Yes				
P_GXE		Asymptotic p-value for this test													Yes				
STAT	Float	Coefficient t-statistic														Yes	Yes	Yes	
NSIG	Int	Total number of SNPs below p-value threshold																Yes	
ISIG	Int	Number of significant																Yes	

		SNPs also passing LD-criterion																	
SNPS	Alphanumeric	List of SNPs in the set																Yes	
UNADJ		Unadjusted p-value																	Yes
GC		Genomic-control corrected p-values																	Yes
BONF		Bonferroni single-step adjusted p-values																	Yes
HOLM		Holm (1979) step-down adjusted p-values																	Yes
SIDAK_SS		Sidak single-step adjusted p-values																	Yes
SIDAK_SD		Sidak step-down adjusted p-values																	Yes
FDR_BH		Benjamini & Hochberg (1995) step-up FDR control																	Yes
FDR_BY		Benjamini & Yekutieli (2001) step-up FDR control																	Yes
NSNP	Int	Number of SNPs in set																Yes	

ANNEXE II

INTERFACE UTILISATEUR ZEPPELIN





ANNEXE III

FICHIER ZEPPELIN-INTERPRETER-SPARK-UBUNTU.LOG

```
 INFO [2017-05-09 20:20:30,309] ({dispatcher-event-loop-0} Logging.scala[logInfo]:54) - Starting task 0.0 in stage 0.0 (TID 0, localhost, partition 0, PROCESS_LOCAL, 5706 bytes)
 INFO [2017-05-09 20:20:30,329] ({dispatcher-event-loop-0} Logging.scala[logInfo]:54) - Starting task 1.0 in stage 0.0 (TID 1, localhost, partition 1, PROCESS_LOCAL, 5706 bytes)
 INFO [2017-05-09 20:20:30,348] ({Executor task launch worker-0} Logging.scala[logInfo]:54) - Running task 0.0 in stage 0.0 (TID 0)
 INFO [2017-05-09 20:20:30,355] ({Executor task launch worker-0} Logging.scala[logInfo]:54) - Fetching file:/home/momo/zeppelin-0.6.2-bin-all/interpreter/spark/pyspark/pyspark.zip with timestamp
1494375580814
 INFO [2017-05-09 20:20:30,356] ({Executor task launch worker-1} Logging.scala[logInfo]:54) - Running task 1.0 in stage 0.0 (TID 1)
 INFO [2017-05-09 20:20:30,659] ({Executor task launch worker-0} Logging.scala[logInfo]:54) - /home/momo/zeppelin-0.6.2-bin-all/interpreter/spark/pyspark/pyspark.zip has been previously copied to /tmp/
spark-42fa2934-6964-4aa3-ae8d-a107ae26270f/userFiles-d324ce5d-f1be-4d9a-ac48-13925ac40559/pyspark.zip
 INFO [2017-05-09 20:20:30,691] ({Executor task launch worker-0} Logging.scala[logInfo]:54) - Fetching file:/home/momo/zeppelin-0.6.2-bin-all/interpreter/spark/pyspark/py4j-0.10.1-src.zip with timestamp
1494375580848
 INFO [2017-05-09 20:20:30,691] ({Executor task launch worker-0} Logging.scala[logInfo]:54) - /home/momo/zeppelin-0.6.2-bin-all/interpreter/spark/pyspark/py4j-0.10.1-src.zip has been previously copied to
tmp/spark-42fa2934-6964-4aa3-ae8d-a107ae26270f/userFiles-d324ce5d-f1be-4d9a-ac48-13925ac40559/py4j-0.10.1-src.zip
 INFO [2017-05-09 20:20:30,779] ({Executor task launch worker-0} Logging.scala[logInfo]:54) - Input split: file:/home/momo/bank/bank-full.csv:0+2305174
 INFO [2017-05-09 20:20:30,785] ({Executor task launch worker-1} Logging.scala[logInfo]:54) - Input split: file:/home/momo/bank/bank-full.csv:2305174+2305174
 INFO [2017-05-09 20:20:30,807] ({Executor task launch worker-0} Configuration.java[warnOnceIfDeprecated]:1009) - mapred.tip.id is deprecated. Instead, use mapreduce.task.id
 INFO [2017-05-09 20:20:30,807] ({Executor task launch worker-0} Configuration.java[warnOnceIfDeprecated]:1009) - mapred.task.id is deprecated. Instead, use mapreduce.task.attempt.id
 INFO [2017-05-09 20:20:30,807] ({Executor task launch worker-0} Configuration.java[warnOnceIfDeprecated]:1009) - mapred.task.is.map is deprecated. Instead, use mapreduce.task.ismap
 INFO [2017-05-09 20:20:30,807] ({Executor task launch worker-0} Configuration.java[warnOnceIfDeprecated]:1009) - mapred.task.partition is deprecated. Instead, use mapreduce.task.partition
 INFO [2017-05-09 20:20:30,807] ({Executor task launch worker-0} Configuration.java[warnOnceIfDeprecated]:1009) - mapred.job.id is deprecated. Instead, use mapreduce.job.id
 INFO [2017-05-09 20:20:30,976] ({Executor task launch worker-0} TransportClientFactory.java[createClient]:250) - Successfully created connection to /192.168.58.133:39695 after 94 ms (0 ms spent in
bootstraps)
 INFO [2017-05-09 20:20:33,414] ({Executor task launch worker-0} Logging.scala[logInfo]:54) - Code generated in 2570.652226 ms
 INFO [2017-05-09 20:20:33,494] ({Executor task launch worker-0} Logging.scala[logInfo]:54) - Code generated in 54.273243 ms
 INFO [2017-05-09 20:20:33,642] ({Executor task launch worker-0} Logging.scala[logInfo]:54) - Code generated in 141.568211 ms
 INFO [2017-05-09 20:20:34,282] ({Executor task launch worker-0} Logging.scala[logInfo]:54) - Code generated in 573.339556 ms
 INFO [2017-05-09 20:20:34,319] ({Executor task launch worker-1} Logging.scala[logInfo]:54) - Code generated in 19.589705 ms
 INFO [2017-05-09 20:20:34,367] ({Executor task launch worker-1} Logging.scala[logInfo]:54) - Code generated in 27.962558 ms
 INFO [2017-05-09 20:20:34,498] ({Executor task launch worker-1} Logging.scala[logInfo]:54) - Code generated in 114.650991 ms
 INFO [2017-05-09 20:20:37,310] ({Executor task launch worker-0} Logging.scala[logInfo]:54) - Finished task 0.0 in stage 0.0 (TID 0). 2224 bytes result sent to driver
 INFO [2017-05-09 20:20:37,329] ({Executor task launch worker-1} Logging.scala[logInfo]:54) - Finished task 1.0 in stage 0.0 (TID 1). 2493 bytes result sent to driver
 INFO [2017-05-09 20:20:37,344] ({task-result-getter-0} Logging.scala[logInfo]:54) - Finished task 0.0 in stage 0.0 (TID 0) in 7093 ms on localhost (1/2)
 INFO [2017-05-09 20:20:37,347] ({task-result-getter-1} Logging.scala[logInfo]:54) - Finished task 1.0 in stage 0.0 (TID 1) in 7024 ms on localhost (2/2)
 INFO [2017-05-09 20:20:37,349] ({dag-scheduler-event-loop} Logging.scala[logInfo]:54) - ShuffleMapStage 0 (take at NativeMethodAccessorImpl.java:-2) finished in 7.104 s
 INFO [2017-05-09 20:20:37,349] ({dag-scheduler-event-loop} Logging.scala[logInfo]:54) - looking for newly runnable stages
 INFO [2017-05-09 20:20:37,349] ({dag-scheduler-event-loop} Logging.scala[logInfo]:54) - running: Set()
 INFO [2017-05-09 20:20:37,350] ({dag-scheduler-event-loop} Logging.scala[logInfo]:54) - waiting: Set(ResultStage 1)
 INFO [2017-05-09 20:20:37,350] ({dag-scheduler-event-loop} Logging.scala[logInfo]:54) - failed: Set()
 INFO [2017-05-09 20:20:37,348] ({task-result-getter-1} Logging.scala[logInfo]:54) - Removed TaskSet 0.0, whose tasks have all completed, from pool default
 INFO [2017-05-09 20:20:37,378] ({dag-scheduler-event-loop} Logging.scala[logInfo]:54) - Submitting ResultStage 1 (MapPartitionsRDD[13] at take at NativeMethodAccessorImpl.java:-2), which has no missing
parents
 INFO [2017-05-09 20:20:37,437] ({dag-scheduler-event-loop} Logging.scala[logInfo]:54) - Block broadcast_2 stored as values in memory (estimated size 25.7 KB, free 408.9 MB)
 INFO [2017-05-09 20:20:37,462] ({dag-scheduler-event-loop} Logging.scala[logInfo]:54) - Block broadcast_2_piece0 stored as bytes in memory (estimated size 12.1 KB, free 408.9 MB)
```

ANNEXE IV

FICHIER ZEPPELIN-UBUNTU.LOG

```
 INFO [2017-05-09 20:18:54,853] ({qtp1273308587-17} InterpreterFactory.java[createInterpretersForNote]:629) - Interpreter org.apache.zeppelin.spark.SparkRInterpreter 1294987669 created
 INFO [2017-05-09 20:18:54,854] ({qtp1273308587-17} InterpreterFactory.java[createInterpretersForNote]:629) - Interpreter org.apache.zeppelin.spark.SparkSqlInterpreter 110990538 created
 INFO [2017-05-09 20:18:54,854] ({qtp1273308587-17} InterpreterFactory.java[createInterpretersForNote]:629) - Interpreter org.apache.zeppelin.spark.DepInterpreter 1402166570 created
 INFO [2017-05-09 20:18:54,869] ({pool-1-thread-2} SchedulerFactory.java[jobStarted]:131) - Job paragraph_1489262737788_-978500461 started by scheduler
org.apache.zeppelin.interpreter.remote.RemoteInterpretershared_session780835379
 INFO [2017-05-09 20:18:54,871] ({pool-1-thread-2} Paragraph.java[jobRun]:252) - run paragraph 20170311-120537_147592620 using null org.apache.zeppelin.interpreter.LazyOpenInterpreter@4bdcd0fa
 INFO [2017-05-09 20:18:54,882] ({pool-1-thread-2} RemoteInterpreterProcess.java[reference]:148) - Run interpreter process [/home/momo/zeppelin-0.6.2-bin-all/bin/interpreter.sh, -d, /home/momo/
zeppelin-0.6.2-bin-all/interpreter/spark, -p, 40932, -l, /home/momo/zeppelin-0.6.2-bin-all/local-repo/2C52ZFXVU]
 INFO [2017-05-09 20:19:02,678] ({pool-1-thread-2} RemoteInterpreter.java[init]:170) - Create remote interpreter org.apache.zeppelin.spark.SparkInterpreter
 INFO [2017-05-09 20:19:03,126] ({pool-1-thread-2} RemoteInterpreter.java[pushAngularObjectRegistryToRemote]:455) - Push local angular object registry from ZeppelinServer to remote interpreter group
2C52ZFXVU:shared_process
 INFO [2017-05-09 20:19:03,276] ({pool-1-thread-2} RemoteInterpreter.java[init]:170) - Create remote interpreter org.apache.zeppelin.spark.PySparkInterpreter
 INFO [2017-05-09 20:19:03,367] ({pool-1-thread-2} RemoteInterpreter.java[init]:170) - Create remote interpreter org.apache.zeppelin.spark.SparkRInterpreter
 INFO [2017-05-09 20:19:03,371] ({pool-1-thread-2} RemoteInterpreter.java[init]:170) - Create remote interpreter org.apache.zeppelin.spark.SparkSqlInterpreter
 INFO [2017-05-09 20:19:03,374] ({pool-1-thread-2} RemoteInterpreter.java[init]:170) - Create remote interpreter org.apache.zeppelin.spark.DepInterpreter
 WARN [2017-05-09 20:19:15,296] ({qtp1273308587-14} SecurityRestApi.java[ticket]:86) - {"status":"OK","message":"","body":{"principal":"anonymous","ticket":"anonymous","roles":"[]"}}
 INFO [2017-05-09 20:19:16,425] ({qtp1273308587-18} NotebookServer.java[onOpen]:97) - New connection from 127.0.0.1 : 39616
 INFO [2017-05-09 20:19:19,882] ({qtp1273308587-13} NotebookServer.java[sendNote]:433) - New operation from 127.0.0.1 : 39616 : anonymous : GET_NOTE : 2C6PKDRD3
 INFO [2017-05-09 20:20:13,958] ({pool-1-thread-2} NotebookServer.java[afterStatusChange]:1150) - Job 20170311-120537_147592620 is finished
 INFO [2017-05-09 20:20:14,097] ({pool-1-thread-2} SchedulerFactory.java[jobFinished]:137) - Job paragraph_1489262737788_-978500461 finished by scheduler
org.apache.zeppelin.interpreter.remote.RemoteInterpretershared_session780835379
 INFO [2017-05-09 20:20:27,270] ({pool-1-thread-2} SchedulerFactory.java[jobStarted]:131) - Job paragraph_1489262780384_-1211011390 started by scheduler
org.apache.zeppelin.interpreter.remote.RemoteInterpretershared_session780835379
 INFO [2017-05-09 20:20:27,280] ({pool-1-thread-2} Paragraph.java[jobRun]:252) - run paragraph 20170311-120620_1901085735 using sql org.apache.zeppelin.interpreter.LazyOpenInterpreter@69d94ca
 INFO [2017-05-09 20:20:30,800] ({qtp1273308587-12} NotebookRestApi.java[getParagraph]:405) - get paragraph 2CARHX5WB 20170311-120620_1901085735
 INFO [2017-05-09 20:20:38,334] ({pool-1-thread-3} SchedulerFactory.java[jobStarted]:131) - Job paragraph_1481650281698_-132736680 started by scheduler
org.apache.zeppelin.interpreter.remote.RemoteInterpretershared_session780835379
 INFO [2017-05-09 20:20:38,349] ({pool-1-thread-3} Paragraph.java[jobRun]:252) - run paragraph 20161213-093121_676309781 using sql org.apache.zeppelin.interpreter.LazyOpenInterpreter@69d94ca
 INFO [2017-05-09 20:20:39,991] ({pool-1-thread-2} NotebookServer.java[afterStatusChange]:1150) - Job 20170311-120620_1901085735 is finished
 INFO [2017-05-09 20:20:40,074] ({pool-1-thread-2} SchedulerFactory.java[jobFinished]:137) - Job paragraph_1489262780384_-1211011390 finished by scheduler
org.apache.zeppelin.interpreter.remote.RemoteInterpretershared_session780835379
 INFO [2017-05-09 20:20:40,817] ({qtp1273308587-16} NotebookRestApi.java[getParagraph]:405) - get paragraph 2C6PKDRD3 20161213-093121_676309781
 INFO [2017-05-09 20:20:42,790] ({pool-1-thread-3} NotebookServer.java[afterStatusChange]:1150) - Job 20161213-093121_676309781 is finished
 INFO [2017-05-09 20:20:42,889] ({pool-1-thread-3} SchedulerFactory.java[jobFinished]:137) - Job paragraph_1481650281698_-132736680 finished by scheduler
org.apache.zeppelin.interpreter.remote.RemoteInterpretershared_session780835379
 INFO [2017-05-09 20:24:10,261] ({pool-1-thread-4} SchedulerFactory.java[jobStarted]:131) - Job paragraph_1489262780384_-1211011390 started by scheduler
org.apache.zeppelin.interpreter.remote.RemoteInterpretershared_session780835379
 INFO [2017-05-09 20:24:10,271] ({pool-1-thread-4} Paragraph.java[jobRun]:252) - run paragraph 20170311-120620_1901085735 using sql org.apache.zeppelin.interpreter.LazyOpenInterpreter@69d94ca
 INFO [2017-05-09 20:24:10,317] ({qtp1273308587-17} NotebookRestApi.java[getParagraph]:405) - get paragraph 2CARHX5WB 20170311-120620_1901085735
 INFO [2017-05-09 20:24:12,592] ({pool-1-thread-4} NotebookServer.java[afterStatusChange]:1150) - Job 20170311-120620_1901085735 is finished
 INFO [2017-05-09 20:24:12,648] ({pool-1-thread-4} SchedulerFactory.java[jobFinished]:137) - Job paragraph_1489262780384_-1211011390 finished by scheduler
org.apache.zeppelin.interpreter.remote.RemoteInterpretershared_session780835379
```

ANNEXE V

WATCHSERVICE

```
/**
 * log file watcher , every time there's an update or create we get
 * notified.
 * @throws IOException
*/
public void startLogFileWatch() throws IOException {
// Folder to be watched
Path dir = Paths.get("/home/momo/zeppelin-0.6.2-bin-all/logs");
// Create a new Watch Service
WatchService watchService = FileSystems.getDefault().newWatchService();
// Register events
dir.register(watchService, StandardWatchEventKinds.ENTRY_CREATE,
        StandardWatchEventKinds.ENTRY_MODIFY);
while (true) {
WatchKey key;
        try {
                // wait for a key to be available
                key = watchService.take();
                } catch (InterruptedException ex) {
                        return;
                }
                for (WatchEvent<?> event : key.pollEvents()) {
                        // get event type
                        WatchEvent.Kind<?> kind = event.kind();
                        // get file name
                        @SuppressWarnings("unchecked")
                        WatchEvent<Path> ev = (WatchEvent<Path>) event;
                        // get the modified file name
                        Path fileName = ev.context();
                        // if created
                        if (kind == StandardWatchEventKinds.ENTRY_CREATE) {
                                        System.out.println("file created: " + fileName);
                        } // if modified
                        else if (kind == StandardWatchEventKinds.ENTRY_MODIFY) {
                                // readFile();
                                }
                        }
                        // IMPORTANT: The key must be reset after processed
                        boolean valid = key.reset();
                        if (!valid) {
                                break;
                        }
                }
                watchService.close();
                System.out.println("Watch service closed.");
}
```

ANNEXE VI

L'INTERFACE ZEPPELIN INTERPRETER

```
/*
 * Licensed to the Apache Software Foundation (ASF) under one or more
 * contributor license agreements.  See the NOTICE file distributed with
 * this work for additional information regarding copyright ownership.
 * The ASF licenses this file to You under the Apache License, Version 2.0
 * (the "License"); you may not use this file except in compliance with
 * the License.  You may obtain a copy of the License at
 *
 *    http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package org.apache.zeppelin.interpreter;
import java.lang.reflect.Field;
import java.net.URL;
import java.util.Arrays;
import java.util.Collections;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Properties;
import org.apache.commons.lang.StringUtils;
import org.apache.commons.lang.reflect.FieldUtils;
import org.apache.zeppelin.annotation.Experimental;
import org.apache.zeppelin.annotation.ZeppelinApi;
import org.apache.zeppelin.interpreter.thrift.InterpreterCompletion;
import org.apache.zeppelin.scheduler.Scheduler;
import org.apache.zeppelin.scheduler.SchedulerFactory;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
/**
 * Interface for interpreters.
 * If you want to implement new Zeppelin interpreter, extend this class
 *
 * Please see,
 * https://zeppelin.apache.org/docs/latest/development/writingzeppelininterpreter.html
 *
 * open(), close(), interpret() is three the most important method you need to implement.
 * cancel(), getProgress(), completion() is good to have
 * getFormType(), getScheduler() determine Zeppelin's behavior
 */
public abstract class Interpreter {
  /**
   * Opens interpreter. You may want to place your initialize routine here.
   * open() is called only once
   */
  @ZeppelinApi
  public abstract void open();
  /**
   * Closes interpreter. You may want to free your resources up here.
   * close() is called only once
   */
  @ZeppelinApi
  public abstract void close();
  /**
   * Run precode if exists.
   */
  @ZeppelinApi
```

```
public InterpreterResult executePrecode(InterpreterContext interpreterContext) {
  String simpleName = this.getClass().getSimpleName();
  String precode = getProperty(String.format("zeppelin.%s.precode", simpleName));
  if (StringUtils.isNotBlank(precode)) {
    return interpret(precode, interpreterContext);
  }
  return null;
}
/**
 * Run code and return result, in synchronous way.
 *
 * @param st statements to run
 */
@ZeppelinApi
public abstract InterpreterResult interpret(String st, InterpreterContext context);
/**
 * Optionally implement the canceling routine to abort interpret() method
 */
@ZeppelinApi
public abstract void cancel(InterpreterContext context);
/**
 * Dynamic form handling
 * see http://zeppelin.apache.org/docs/dynamicform.html
 *
 * @return FormType.SIMPLE enables simple pattern replacement (eg. Hello ${name=world}),
 * FormType.NATIVE handles form in API
 */
@ZeppelinApi
public abstract FormType getFormType();
/**
 * get interpret() method running process in percentage.
 *
 * @return number between 0-100
 */
@ZeppelinApi
public abstract int getProgress(InterpreterContext context);
/**
 * Get completion list based on cursor position.
 * By implementing this method, it enables auto-completion.
 *
 * @param buf statements
 * @param cursor cursor position in statements
 * @param interpreterContext
 * @return list of possible completion. Return empty list if there're nothing to return.
 */
@ZeppelinApi
public List<InterpreterCompletion> completion(String buf, int cursor,
    InterpreterContext interpreterContext)  {
  return null;
}
/**
 * Interpreter can implements it's own scheduler by overriding this method.
 * There're two default scheduler provided, FIFO, Parallel.
 * If your interpret() can handle concurrent request, use Parallel or use FIFO.
 *
 * You can get default scheduler by using
 * SchedulerFactory.singleton().createOrGetFIFOScheduler()
 * SchedulerFactory.singleton().createOrGetParallelScheduler()
 *
 * @return return scheduler instance. This method can be called multiple times and have to return
 * the same instance. Can not return null.
 */
@ZeppelinApi
public Scheduler getScheduler() {
  return SchedulerFactory.singleton().createOrGetFIFOScheduler("interpreter_" + this.hashCode());
}
public static Logger logger = LoggerFactory.getLogger(Interpreter.class);
private InterpreterGroup interpreterGroup;
private URL[] classloaderUrls;
protected Properties property;
private String userName;
```

```
@ZeppelinApi
public Interpreter(Properties property) {
  logger.debug("Properties: {}", property);
  this.property = property;
}
public void setProperty(Properties property) {
  this.property = property;
}
@ZeppelinApi
public Properties getProperty() {
  Properties p = new Properties();
  p.putAll(property);
  RegisteredInterpreter registeredInterpreter = Interpreter.findRegisteredInterpreterByClassName(
      getClassName());
  if (null != registeredInterpreter) {
    Map<String, InterpreterProperty> defaultProperties = registeredInterpreter.getProperties();
    for (String k : defaultProperties.keySet()) {
      if (!p.containsKey(k)) {
        String value = defaultProperties.get(k).getValue();
        if (value != null) {
          p.put(k, defaultProperties.get(k).getValue());
        }
      }
    }
  }
  replaceContextParameters(p);
  return p;
}
@ZeppelinApi
public String getProperty(String key) {
  logger.debug("key: {}, value: {}", key, getProperty().getProperty(key));
  return getProperty().getProperty(key);
}
public String getClassName() {
  return this.getClass().getName();
}
public void setUserName(String userName) {
  this.userName = userName;
}
public String getUserName() {
  return this.userName;
}
public void setInterpreterGroup(InterpreterGroup interpreterGroup) {
  this.interpreterGroup = interpreterGroup;
}
@ZeppelinApi
public InterpreterGroup getInterpreterGroup() {
  return this.interpreterGroup;
}
public URL[] getClassloaderUrls() {
  return classloaderUrls;
}
public void setClassloaderUrls(URL[] classloaderUrls) {
  this.classloaderUrls = classloaderUrls;
}
/**
 * General function to register hook event
 *
 * @param noteId - Note to bind hook to
 * @param event The type of event to hook to (pre_exec, post_exec)
 * @param cmd The code to be executed by the interpreter on given event
 */
@Experimental
public void registerHook(String noteId, String event, String cmd) {
  InterpreterHookRegistry hooks = interpreterGroup.getInterpreterHookRegistry();
  String className = getClassName();
  hooks.register(noteId, className, event, cmd);
}
/**
 * registerHook() wrapper for global scope
 *
```

```
 * @param event The type of event to hook to (pre_exec, post_exec)
 * @param cmd The code to be executed by the interpreter on given event
 */
@Experimental
public void registerHook(String event, String cmd) {
  registerHook(null, event, cmd);
}
/**
 * Get the hook code
 *
 * @param noteId - Note to bind hook to
 * @param event The type of event to hook to (pre_exec, post_exec)
 */
@Experimental
public String getHook(String noteId, String event) {
  InterpreterHookRegistry hooks = interpreterGroup.getInterpreterHookRegistry();
  String className = getClassName();
  return hooks.get(noteId, className, event);
}
/**
 * getHook() wrapper for global scope
 *
 * @param event The type of event to hook to (pre_exec, post_exec)
 */
@Experimental
public String getHook(String event) {
  return getHook(null, event);
}
/**
 * Unbind code from given hook event
 *
 * @param noteId - Note to bind hook to
 * @param event The type of event to hook to (pre_exec, post_exec)
 */
@Experimental
public void unregisterHook(String noteId, String event) {
  InterpreterHookRegistry hooks = interpreterGroup.getInterpreterHookRegistry();
  String className = getClassName();
  hooks.unregister(noteId, className, event);
}
/**
 * unregisterHook() wrapper for global scope
 *
 * @param event The type of event to hook to (pre_exec, post_exec)
 */
@Experimental
public void unregisterHook(String event) {
  unregisterHook(null, event);
}
@ZeppelinApi
public Interpreter getInterpreterInTheSameSessionByClassName(String className) {
  synchronized (interpreterGroup) {
    for (List<Interpreter> interpreters : interpreterGroup.values()) {
      boolean belongsToSameNoteGroup = false;
      Interpreter interpreterFound = null;
      for (Interpreter intp : interpreters) {
        if (intp.getClassName().equals(className)) {
          interpreterFound = intp;
        }
        Interpreter p = intp;
        while (p instanceof WrappedInterpreter) {
          p = ((WrappedInterpreter) p).getInnerInterpreter();
        }
        if (this == p) {
          belongsToSameNoteGroup = true;
        }
      }
      if (belongsToSameNoteGroup) {
        return interpreterFound;
      }
    }
```

```
  }
  return null;
}
/**
 * Replace markers #{contextFieldName} by values from {@link InterpreterContext} fields
 * with same name and marker #{user}. If value == null then replace by empty string.
 */
private void replaceContextParameters(Properties properties) {
  InterpreterContext interpreterContext = InterpreterContext.get();
  if (interpreterContext != null) {
    String markerTemplate = "#\\{%s\\}";
    List<String> skipFields = Arrays.asList("paragraphTitle", "paragraphId", "paragraphText");
    List typesToProcess = Arrays.asList(String.class, Double.class, Float.class, Short.class,
        Byte.class, Character.class, Boolean.class, Integer.class, Long.class);
    for (String key : properties.stringPropertyNames()) {
      String p = properties.getProperty(key);
      if (StringUtils.isNotEmpty(p)) {
        for (Field field : InterpreterContext.class.getDeclaredFields()) {
          Class clazz = field.getType();
          if (!skipFields.contains(field.getName()) && (typesToProcess.contains(clazz)
              || clazz.isPrimitive())) {
            Object value = null;
            try {
              value = FieldUtils.readField(field, interpreterContext, true);
            } catch (Exception e) {
              logger.error("Cannot read value of field {0}", field.getName());
            }
            p = p.replaceAll(String.format(markerTemplate, field.getName()),
                value != null ? value.toString() : StringUtils.EMPTY);
          }
        }
        p = p.replaceAll(String.format(markerTemplate, "user"),
            StringUtils.defaultString(userName, StringUtils.EMPTY));
        properties.setProperty(key, p);
      }
    }
  }
}
/**
 * Type of interpreter.
 */
public static enum FormType {
  NATIVE, SIMPLE, NONE
}
/**
 * Represent registered interpreter class
 */
public static class RegisteredInterpreter {
  private String group;
  private String name;
  private String className;
  private boolean defaultInterpreter;
  private Map<String, InterpreterProperty> properties;
  private Map<String, Object> editor;
  private String path;
  private InterpreterOption option;
  private InterpreterRunner runner;
  public RegisteredInterpreter(String name, String group, String className,
      Map<String, InterpreterProperty> properties) {
    this(name, group, className, false, properties);
  }
  public RegisteredInterpreter(String name, String group, String className,
      boolean defaultInterpreter, Map<String, InterpreterProperty> properties) {
    super();
    this.name = name;
    this.group = group;
    this.className = className;
    this.defaultInterpreter = defaultInterpreter;
    this.properties = properties;
    this.editor = new HashMap<>();
  }
```

```
  public String getName() {
    return name;
  }
  public String getGroup() {
    return group;
  }
  public String getClassName() {
    return className;
  }
  public boolean isDefaultInterpreter() {
    return defaultInterpreter;
  }
  public void setDefaultInterpreter(boolean defaultInterpreter) {
    this.defaultInterpreter = defaultInterpreter;
  }
  public Map<String, InterpreterProperty> getProperties() {
    return properties;
  }
  public Map<String, Object> getEditor() {
    return editor;
  }
  public void setPath(String path) {
    this.path = path;
  }
  public String getPath() {
    return path;
  }
  public String getInterpreterKey() {
    return getGroup() + "." + getName();
  }
  public InterpreterOption getOption() {
    return option;
  }
  public InterpreterRunner getRunner() {
    return runner;
  }
}
/**
 * Type of Scheduling.
 */
public static enum SchedulingMode {
  FIFO, PARALLEL
}
public static Map<String, RegisteredInterpreter> registeredInterpreters = Collections
    .synchronizedMap(new HashMap<String, RegisteredInterpreter>());
@Deprecated
public static void register(String name, String group, String className,
    Map<String, InterpreterProperty> properties) {
  register(name, group, className, false, properties);
}
@Deprecated
public static void register(String name, String group, String className,
    boolean defaultInterpreter, Map<String, InterpreterProperty> properties) {
  logger.warn("Static initialization is deprecated for interpreter {}, You should change it " +
      "to use interpreter-setting.json in your jar or " +
      "interpreter/{interpreter}/interpreter-setting.json", name);
  register(new RegisteredInterpreter(name, group, className, defaultInterpreter, properties));
}
@Deprecated
public static void register(RegisteredInterpreter registeredInterpreter) {
  String interpreterKey = registeredInterpreter.getInterpreterKey();
  if (!registeredInterpreters.containsKey(interpreterKey)) {
    registeredInterpreters.put(interpreterKey, registeredInterpreter);
  } else {
    RegisteredInterpreter existInterpreter = registeredInterpreters.get(interpreterKey);
    if (!existInterpreter.getProperties().equals(registeredInterpreter.getProperties())) {
      logger.error("exist registeredInterpreter with the same key but has different settings.");
    }
  }
}
public static RegisteredInterpreter findRegisteredInterpreterByClassName(String className) {
```

```
    for (RegisteredInterpreter ri : registeredInterpreters.values()) {
      if (ri.getClassName().equals(className)) {
        return ri;
      }
    }
    return null;
  }
}
```

ANNEXE VII

INSTALLATION DES OUTILS

1) Installation de PostgreSQL
    a. sudo apt-get install postgresql
2) Création de la base de donnée « zeppelindb »
    a. sudo -su postgres
    b. psql -c "CREATE USER zeppelin WITH PASSWORD 'zeppelin';"
    c. psql -c "CREATE DATABASE zeppelindb OWNER zeppelin;"
    d. CREATE TABLE data (
        notebookid text,
        notebookname text,
        paragraphid text,
        code text,
        output text,
        input text,
        datecreated text,
        dateupdated text,
        datestarted text);
    e. GRANT ALL PRIVILEGES ON TABLE data TO zeppelin;

3) Installation de Zeppelin
    a. télécharger le fichier « zeppelin-0.6.2-bin-all.tgz »
    b. extraire le fichier dans un répertoire à votre choix
4) Démarrage de Zeppelin
    a. cd zeppelin-0.6.2-bin-all/bin/
    b. ./zeppelin-daemon.sh start
    c. insérer « localhost:8090 » dans le navigateur pour accéder au UI de Zeppelin

   d. insérer « http://hadoop:4040/jobs/» dans le navigateur pour accéder au UI de Spark

5) Arrêt de Zeppelin
   a. cd zeppelin-0.6.2-bin-all/bin/
   b. ./zeppelin-daemon.sh stop

6) Installer Java
   a. sudo apt-get install oracle-java8-installer

BIBLIOGRAPHIE

[1] Wikipedia Foundation, Inc. 2015. Bio-informatique. En ligne. <https://fr.wikipedia.org/wiki/Bio-informatique>. Consulté le 15 Janvier 2016.

[2] Wikipedia Foundation, Inc. 2015.Modèle de Markov caché. En ligne. <https://fr.wikipedia.org/wiki/Mod%C3%A8le_de_Markov_cach%C3%A9>. Consulté le 30 janvier 2016.

[3] National Center for Biotechnology Information, U.S. National Library of Medicine. 2005. En ligne. <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1310647/ >. Consulté le 20 février 2016.

[4] The International Genome Sample Resource. 2016. En ligne. <http://www.internationalgenome.org>. Consulté le 1er mars 2016.

[5] Mikael Falconnet. 2013. Chaînes de Markov analyse de séquences biologiques.L3 Génie Biologique et Informatique . pp. 15-18 .

[6] National Center for Biotechnology Information, U.S. National Library of Medicine. 2005. En ligne. <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2766791/>. Consulté le 3 mars 2016.

[7] Wikipedia Foundation, Inc. 2015. Locus. En ligne. <https://fr.wikipedia.org/wiki/Locus>. Consulté le 30 janvier 2016.

[8] Wikipedia Foundation, Inc. 2017. Projet 1000 Genomes. En ligne. <https://fr.wikipedia.org/wiki/Projet_1000_Genomes>. Consulté le 10 février 2017.

[9] Golden Helix, Inc. 2015. Comparing Beagle, Impute2, and MiniMac imputation methods for accuracy computation time and memory usage. En ligne. <http://blog.goldenhelix.com/alaughbaum/comparing-beagle-impute2-and-minimac-imputation-methods-for-accuracy-computation-time-and-memory-usage/>. Consulté le 5 juillet 2016.

[10] PLOS, Corporation. 2009. A Flexible and Accurate Imputation Method for the Next Generation of GWAS. En ligne. <http://journals.plos.org/plosgenetics/article?id=10.1371/journal.pgen.1000529#s3> . Consulté le 5 juillet 2016.

[11] National Center for Biotechnology Information, U.S. National Library of Medicine. 2010. A novel method for haplotype clustering and visualization. En ligne. <https://www.ncbi.nlm.nih.gov/pubmed/19479748> . Consulté le 7 juillet 2016.

[12] Université Pierre & Marie Curie. La science à Paris:Génétique. En ligne. <http://www.edu.upmc.fr/sdv/masselot_05001/polymorphisme/snp.html>. Consulté le 18 février 2017.

[13] Wikipedia Foundation, Inc. 2015. Locus. En ligne. <https://en.wikipedia.org/wiki/Genome-wide_association_study> . Consulté le 30 janvier 2016.

[14] National Center for Biotechnology Information, U.S. National Library of Medicine. 2012. Genome-Wide Association Studies. En ligne. <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3531285/>. Consulté le 18 février 2017.

[15] University of Michigan, Center for Statistical Genetics. 2016. MACH. En ligne <http://csg.sph.umich.edu/abecasis/mach/index.html>. Consulté le 10 juillet 2016.

[16] Impute2. 2014. Impute. En ligne. <https://mathgen.stats.ox.ac.uk/impute/impute_v2.html#home>. Consulté le 10 juillet 2016.

[17] OMIC Tools, Genotype Imputation. 2015. Genotype Imputation Software Tools GWAS. En ligne. <https://omictools.com/genotype-imputation-category> . Consulté le 10 juillet 2016.

[18] BEAGLE. 2017. Version 4.1. En ligne. <http://faculty.washington.edu/browning/beagle/beagle.html>. Consulté le 10 février 2017.

[19] Wikipedia Foundation, Inc. 2016. Homozygote. En ligne. <https://fr.wikipedia.org/wiki/Homozygote> . Consulté le 15 février 2017.

[20] Paul Scheet et Mathew Stephens. 2008. Documentation for fastPHASE 1.4*. En ligne. <http://scheet.org/code/fastphase_doc_1.4.pdf> . Consulté le 10 juillet 2016.

[21] ampLab, UC Berkely. 2008. ADAM: Genomics Formats and Processing Patterns of Cloud ScaleComputing. En ligne.

<https://translate.google.ca/translate?hl=fr&sl=en&tl=fr&u=https%3A%2F%2Fampla b.cs.berkeley.edu%2Fpublication%2Fadam-genomics-formats-and-processing-patterns-for-cloud-scale-computing%2F&anno=2>. Consulté le 15 février 2017.

[22] SNPTEST.[s.d]. SNPTEST v2.5.2. En ligne. <https://mathgen.stats.ox.ac.uk/genetics_software/snptest/snptest.html#introduction>. Consulté le 15 février 2017.

[23] PLINK. 2016. En ligne. <https://www.cog-genomics.org/plink2> . Consulté le 5 février 2016.

[24] GWAS file format. [s.d]. En ligne. <http://www.stats.ox.ac.uk/~marchini/software/gwas/file_format.html#Genotype_File_Format>. Consulté le 22 février 2017.

[25] The Variant Call Format. 2016. Version 4.2 Specification. En ligne. <http://samtools.github.io/hts-specs/VCFv4.2.pdf> . Consulté le 22 février 2017.

[26] Genotypes, Phenotypes and Markers-Definitions. 2017. En ligne. <http://www.sigmaaldrich.com/life-science/molecular-biology/cloning-and-expression/learning-center/definitions.html>. Consulté le 22 février 2017.

[27] Diego Alvarez, Le Big Data dans le domaine de la génomique, projet de maitrise appliqué de 9 crédits en Génie Logiciel à l' École de Technologie Supérieure, Décembre 2016, 84 p. En ligne. <http://publicationslist.org/data/a.april/ref-570/Rapport_Diego_Alvarez_final.pdf.> . Consulté le 26 février 2017.

[28] Apache Zeppelin. 2017. En ligne. < https://zeppelin.apache.org/ >. Consulté le 10 janvier 2017.

[29] Apache Zeppelin 0.6.2. 2017. Quick Start . En ligne. <http://zeppelin.apache.org/docs/0.6.2/install/install.html>. Consulté le 10 janvier 2017.

[30] Apache Software Foundation. 2017. Apache HttpComponents. En ligne. <https://hc.apache.org/ >. Consulté le 15 janvier 2017.

[31] PostgreSQL. 2017. The PostgreSQL Global Development Group. En ligne. <https://www.postgresql.org/ >. Consulté le 15 janvier 2017.

[32] Apache Spark. 2017. Lightning-fast computing cluster. En ligne.
<http://spark.apache.org/ >. Consulté le 15 janvier 2017.

[33] Apache Zeppelin 0.6.2. 2017. Writing a New Interpreter . En ligne.
<http://zeppelin.apache.org/docs/0.6.2/development/writingzeppelininterpreter.html>.
Consulté le 20 janvier 2017.

[34]Github.[s.d].En ligne.<https://github.com/apache/zeppelin/blob/master/zeppelin-interpreter/src/main/java/org/apache/zeppelin/interpreter/Interpreter.java#L207,L218>.
Consulté le 22 janvier 2017.