ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

RAPPORT DE PROJET PRÉSENTÉ À
L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

COMME EXIGENCE PARTIELLE
À L'OBTENTION DE MAITRISE EN GENIE LOGICIEL

PAR
Diego ALVAREZ

LE BIG DATA DANS LE DOMAINE DE LA GÉNOMIQUE

MONTRÉAL, LE 16 DECEMBRE 2016

PRÉSENTATION DU JURY

CE RAPPORT DE PROJET A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE :

Professeur Alain April, directeur du 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

Pour m'avoir permis de faire partie de l'équipe du projet QnGene au CRCHUM, pour leur aide, leur temps et leur support, je tiens à remercier :

Mme. Marisa DURAN, mon épouse : pour son encouragement constant à ne pas abandonner, pour les efforts et sacrifices qu'elle a fait dans notre vie quotidienne afin de que je puisse finir mes études. Sans elle, et la joie de mes enfants, je n'aurais même pas pu finir le premier cours de la maitrise.

Professeur Alain APRIL, directeur du projet à l'École de Technologie Supérieure : pour avoir accepté de me diriger dans ce projet, pour ses judicieux conseils et pour m'avoir encadré dans le travail.

M. David LAUZON, étudiant de doctorat et responsable de la conception Big Data du projet QnGene au CRCHUM : pour sa patience et pour trouver constamment des manières pour vulgariser le domaine de la génomique.

Mme. Béatriz KANZKI, bio-informaticienne au CRCHUM : pour son support tout au long du projet.

LE BIG DATA DANS LE DOMAINE DE LA GÉNOMIQUE

Diego ALVAREZ

RÉSUMÉ

Ce rapport de projet appliqué de 9 crédits, à la maîtrise en génie logiciel, a pour but de présenter la démarche effectuée afin de traiter et d'adapter, au format ADAM de l'Université de Californie à Berkeley, les génotypes imputés provenant du logiciel d'imputation IMPUTE2 (et d'autre logiciel générant le même format de fichiers) et de les intégrer aux données des bases de données ADVANCE et dbSNP.

Le projet libre ADAM, originaire de l'Université de Californie à Berkeley, est un ensemble de formats de fichiers, interfaces et outils conçus pour standardiser le format des données et permettre la mise à l'échelle du traitement des données massives présentes dans le domaine de la génomique.

Le logiciel IMPUTE2 (c.-à-d. IMPUTE version 2), développé par l'Université d'Oxford, est un programme d'imputation des génotypes (c.-à-d. d'inférence statistique de génotypes non observés) basé sur des haplotypes observés. Ce logiciel génère deux formats de fichiers, les « .gen », qui contiennent les probabilités d'occurrence d'un génotype donné pour chaque individu étudié, et le « .sample », qui contient l'information des individus étudiés.

L'objectif de mon projet est d'étendre celui de l'étudiant Simon Grondin, en documentant les prérequis et activités nécessaires pour reproduire son projet et en effectuant les étapes nécessaires pour générer un nouveau jeu de données contenant les génotypes imputés d'IMPUTE2 ainsi que les données cliniques des patients d'ADVANCE en format ADAM.

BIG DATA IN THE FIELD OF GENOMICS

Diego ALVAREZ

ABSTRACT

The purpose of this 9 credits applied project report, at the software engineering master program, presents the approach taken in order to process and adapt the imputed genotypes, from the IMPUTE2 imputation software (or any other software using the same file format) to the ADAM format. Additionally this data needed to be integrated with the data of the ADVANCE and dbSNP databases.

Berkeley's ADAM project, issue from a UC Berkeley initiative, proposes a new normalized file format, APIs and tools designed to address scalability issues arising from the current genomic file formats.

The University of Oxford's IMPUTE2 (IMPUTE version 2) is a genotype imputation program (statistical inference of unobserved genotypes) based on observed haplotypes which generates two file formats, « .gen », with the studied individual's genotype probabilities, and « .sample », with the individuals information.

The project's objective is to extend the project of the student Simon Grondin by documenting the pre-requirements and necessary activities to reproduce it and carrying out all the necessary steps to generate a new dataset in ADAM's format, containing the imputed genotypes of IMPUTE2 as well as the clinical data of ADVANCE patients.

TABLE DES MATIÉRES

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

ADVANCE: acronyme en anglais de *Action in diabetes and vascular disease*.

CRCHUM: Centre de Recherche du Centre Hospitalier Universitaire de Montréal

dbSNP: acronyme en anglais de *Single Nucleotide Polymorphism database*

ÉTS: École de technologie supérieure

GWAS: Genome-Wide Association Study

SNP: Single-Nucleotide Polymorphism

T2D : Type 2 diabetes.

.tsv: Tab-Separeted Values

LISTE DES DEFINITIONS

ADAM: c'est un ensemble de formats de fichiers et APIs pour le traitement de données génomiques. C'est un cadre de travail (c.-à-d. un *framework*) utilisant la technologie Apache Spark, les schémas d'Apache Avro et le format de stockage des données Apache Parquet.

ADVANCE: base de données d'études cliniques qui contient de l'information sur les patients, les tests et leur génotype.

ADVANCE-to-ADAM (A2A): nom simplifié du projet de Simon Grondin.

Génotype: c'est l'ensemble des gènes d'un individu.

Haplotype: c'est un ensemble de gènes situés côte à côte sur un chromosome.

Oxford-to-ADAM(O2A) : nom simplifié du projet d'adaptation du format ADAM pour y ajouter les génotypes imputés.

Phénotype: c'est l'ensemble de caractéristiques observables ou détectables d'un organisme.

Pré-ADAM: nom simplifié des fichiers « .tsv » générés par le script export.py qui seront ensuite utilisés pour générer les fichiers Parquet-ADAM.

QnGene CRCHUM: nom simplifié du projet génomique Big Data de David Lauzon (c.-à-d. son PhD), supervisé par le professeur Alain April.

Suite Oxford: groupe de logiciels conçus par l'Université d' Oxford. Ce groupe est composé par les logiciels IMPUTE2, CHIAMO, HAPGEN, SNPTEST et GTOOL.

Tas : la mémoire tas, ou *heap* en anglais, est un segment de mémoire logique dans la mémoire de l'ordinateur que les langages de programmation réservent, lors de l'exécution d'un programme donné, pour faire ses calculs internes et l'stockage de données.

INTRODUCTION

Le domaine du traitement des données génétiques, probablement en raison de sa nouveauté, est méconnu. C'est donc un domaine encore très jeune, couteux et complexe. L'obtention des données et leurs manipulations sont faites par plusieurs fournisseurs et acteurs et sa maturité technologique est faible.

Les différentes ententes entre les universités, centres de recherche et laboratoires ainsi qu'une participation accrue des gouvernements, au cours des dernières années, a fait augmenter la disponibilité de données génétiques accessibles aux chercheurs du monde entier. Les gouvernements ont pris conscience du potentiel de l'étude de la génomique dans le domaine de la santé et ont commencé à investir des sommes considérables dans la recherche, la découverte de gènes et leurs impacts potentiels sur les traitements futurs de la santé. La détection précoce de certains types de maladies et le diagnostic des causes reliées à la génétique du patient vont influencer les médicaments du futur et proposent une meilleure qualité de vie pour les patients.

L'apparition de logiciels libres récents, diversifiés et ouverts à la communauté scientifique accélère aussi les découvertes dans ce domaine. Cela contribue à la réduction des couts et à la contribution et participation des chercheurs d'autres domaines, tels que le génie logiciel, le « computer science » et les mathématiques, qui contribuent de plus en plus au domaine. Les technologies émergentes, du domaine du Big Data pourraient jouer un rôle central, car la capacité de traiter des quantités massives de données, et ce à très grande vitesse et à une fraction du coût offre la possibilité à des plus petits acteurs de contribuer.

Dans le but de contribuer à la recherche des causes génétiques des complications du diabète de type 2, et de devenir une référence dans le domaine du Big Data appliqué à la génomique, le professeur Alain April de l'ÉTS, avec la collaboration d'étudiants de maitrise et de doctorat, font actuellement une expérimentation au CRCHUM, qui utilise des technologies traditionnelles (c.-à-d. scripts et procédures) non adaptées pour le traitement efficace de

quantités massives de données génomiques. L'objectif du projet est de migrer ces technologies vers des technologies Big Data. Un prérequis, pour l'utilisation de ces nouvelles technologies, est l'adaptation et la conversion de différentes structures de données existantes, incluant les génotypes imputés, vers le nouveau format de données proposé par le projet ADAM de l'Université Berkeley.

CHAPITRE 1

CONTEXTE

Ce chapitre introduit l'équipe du projet QnGene au CRCHUM, dont je fais partie. Les objectifs du projet et les sources de données à utiliser. Une présentation sommaire du projet de Simon Grondin est présentée ainsi que le besoin du client qu'a déclenché ce projet.

1.1 Présentation du projet et de l'équipe QnGene au CRCHUM

L'équipe du projet QnGene du CRCHUM est composée par le directeur du projet, le professeur Alain April et de ses étudiants de maitrise et de doctorat en génie logiciel. L'équipe a pour but de migrer les différentes sources de données, en assurer leur qualité et les adapter au format ADAM [1], de l'université Berkeley, afin de pouvoir faire une mise en échelle permettant d'effectuer des analyses de données génétiques, à grande échelles, très rapidement et interactivement.

En raison du volume actuel des données à traiter (environ 175 Go), l'hétérogénéité des sources de données et les temps de réponse visés pour le traitement de ces données (c.-à-d. pas plus des 4 heures), une solution Big Data a été proposée au tout début du projet. Le projet a été conçu d'une manière modulaire afin de permettre aux différents membres de l'équipe, et à ceux qui se sont ajoutés tout au long du projet de travailler sans avoir de dépendances entre les différentes parties.

1.2 Le mandat de ce projet appliqué

La définition de cette partie du projet de recherche a subi certains ajustements pendant sa réalisation. Il a débuté par l'analyse des algorithmes GWAS, suivi de l'analyse du modèle de données de la base de données CARTaGene. Finalement il s'est concentré sur l'adaptation des fichiers générés par le logiciel IMPUTE2 (c.-à-d. IMPUTE version 2), développé par

l'Université d'Oxford, vers le format du projet ADAM de l'Université de Californie Berkeley. Ces ajustements proviennent du raffinement progressif des besoins du CRCHUM, une meilleure compréhension du domaine d'affaires et au roulement du personnel impliqué dans le projet.

L'objectif du projet de recherche appliquée de 9 crédits a été:

1. Reproduire et documenter les étapes pour l'adaptation d'ADVANCE et dbSNP en format ADAM, initié par l'étudiant Simon Grondin (A2A). Point de départ pour la deuxième partie (O2A) (voir ANNEXE « Liste de Prérequis »).
2. Concevoir et développer l'adaptation des fichiers « .gen » et « .sample » d'IMPUTE2 au format ADAM (O2A).

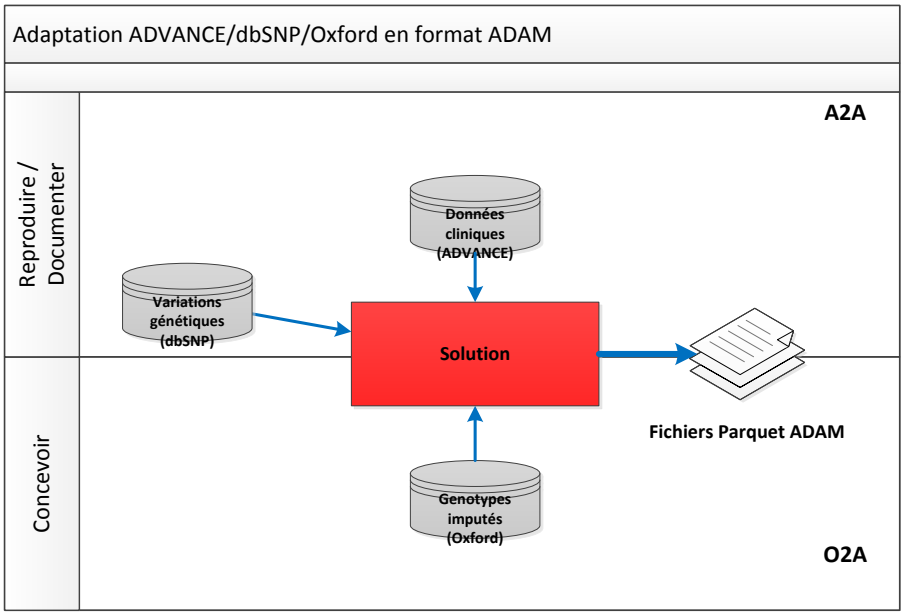


Figure 1-1 Objectifs du projet de recherche appliquée

La figure 1.1 présente les sources de données et le résultat attendu à la fin du processus d'adaptation. La partie d'A2A devait être document de manière de permettre la reproduction du processus d'adaptation par un usager non familiarisé avec la solution. La conception et

développement de la partie O2A a demandé la modification de certains scripts de la partie A2A et la création de nouveaux scripts.

1.3 Le projet de l'étudiant Simon Grondin

L'objectif du projet de Simon Grondin [3], étudiant du baccalauréat en génie logiciel à l'ÉTS et ancien membre de l'équipe QnGene au CRCHUM, était d'exporter les bases de données ADVANCE et dbSNP, actuellement en format de fichiers « *dump* » et « .vcf », vers un état intermédiaire basé sur des fichiers « .tsv ». Ensuite, réécrire ces fichiers intermédiaires dans le format Parquet afin de pouvoir les lire et les interroger avec la technologie Spark SQL.

1.4 Analyse des exigences

Suite à plusieurs rencontres d'équipe, effectuées à la conclusion d'un jalon de projet, c'est-à-dire celui de la migration des données cliniques et des génotypes de la base de données ADVANCE (A2A), le responsable du projet s'est rendu compte que les génotypes n'étaient pas utilisés par les bio-informaticiens du CRCHUM. Il avait été mal informé et a découvert que leurs analyses actuelles utilisaient plutôt les génotypes imputés à partir des fichiers de format « .gen » provenant de la suite de logiciels d'Oxford (IMPUTE2, SNPTEST, et bien d'autres). Sans ces données imputées, les données d'ADVANCE n'auraient pas une grande utilité pour les chercheurs.

Afin de résoudre ce problème, les bio-informaticiens du CRCHUM ont fourni des fichiers contenant les génotypes imputés d'un sous-ensemble d'individus de la BD ADVANCE. L'exigence de ce projet est d'intégrer les génotypes imputés des fichiers d'Oxford aux individus de la base de données ADVANCE et adapter le tout au nouveau format étendu ADAM.

1.5 Présentation du contenu de chaque base de données

1.5.1 La base de données dbSNP

La base de données dbSNP [4], provenant du NCBI (National Center for Biotechnology Information) est une base de données de variations génétiques, ou SNP (single nucléotide polymorphismes). La *dbSNP human build 142*, la version utilisée dans ce projet, contient 112 millions de SNP de référence.

1.5.2 La base de données ADVANCE

La base ADVANCE [5] de données est le résultat d'une étude clinique incluant 215 centres collaborateurs dans 20 pays d'Asie, d'Australie, d'Europe et d'Amérique du Nord. L'étude a été conçue pour randomiser plus de 10 000 patients atteints de diabète de type 2 soumis à quatre catégories de traitements : 1) l'abaissement intensif du glucose (y compris le gliclazide MR) et l'abaissement supplémentaire de la PA de routine (perindopril / indapamide Combinaison), 2) la glycémie standard et la diminution systématique de la PA, 3) l'hypoglycémie intensive et le placebo; et finalement 4) la thérapie de glucose standard et placebo (6). La version utilisée incluait les données de 5157 patients.

1.5.3 IMPUTE2 « .gen »/ « .sample »

Les fichiers de données générés par le logiciel IMPUTE2 [2] contiennent les génotypes imputés d'un sous-ensemble des individus d'ADVANCE. Pour ce projet, ils contiennent 3409 individus provenant du SNP chip V.6.0 [33].

1.5.4 Conclusion

Ce chapitre a présenté les objectifs de mon projet de recherche appliquée, un sommaire du projet de Simon Grondin, et la composition de l'équipe QnGene au CRCHUM. En outre, j'ai présenté le contenu des sources de données que j'ai dû gérer, des sources hétérogènes, mais

complémentaires qui vont permettre aux bio-informaticiens d'effectuer, de manière simple, des analyses poussées et diversifiées.

Le chapitre suivant présente la structure et contenu du jeu de données d'IMPUTE2.

CHAPITRE 2

LE JEU DE DONNÉES DE L'OUTIL IMPUTE2 DE OXFORD

Les fichiers « .gen » et « .sample » ont été conçus pour être utilisés par les applications IMPUTE2 [2], CHIAMO [6], HAPGEN [7], SNPTEST [8] et GTOOL [9]. Ces applications ont toutes été créées par l'Université d'Oxford, pour des objectifs différents, mais qui utilisent toutes un format particulier. Ces applications sont toutes capables de lire et/ou générer des fichiers de format « .gen » et « .sample », à l'exception de la version 2 du logiciel SNPTEST qui inclue quelques petites modifications de format. Ces formats de fichiers sont utilisés lors d'études d'association pangénomique, ou mieux connue sous le nom de GWAS [10].

2.1 Le fichier de format « .sample »

Ce fichier contient l'information sur chaque individu. L'information est disposée sur une ligne de ce fichier et les informations sur chaque individu se retrouvent sur des colonnes différentes. Les champs de données des individus, disponibles dans ce fichier sont : l'ID, l'ID de famille (ou 2e ID), les covariables et les phénotypes. Les 3 premières colonnes sont obligatoires, le nombre de colonnes peut varier. Le fichier de format « .sample » possède 3 parties (voir l'exemple de la figure 2.1) 1) le nom de colonnes, dans la 1re ligne du fichier; 2) le type de donnée, dans la 2e ligne du fichier; et 3) la valeur pour chaque colonne, à partir de la 3e ligne du fichier.

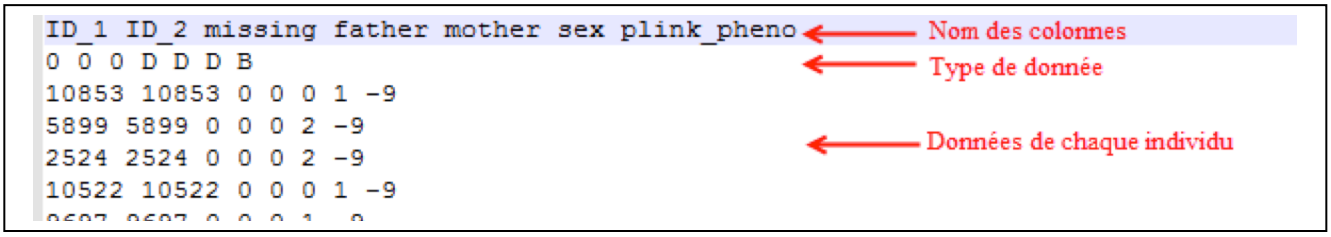


Figure 2-1 Le format de fichier « .sample »

2.2 Le fichier de format « .gen »

Ces fichiers contiennent les génotypes imputés des individus. Chaque variation SNP est localisée sur une ligne diffèrent du fichier et les probabilités pour chaque individu sont localisées dans des colonnes différentes. Les probabilités des individus sont représentées par groupe de 3 colonnes, et chaque individu est identifié par sa position dans le fichier. Les individus dans le format « .gen » n'ont pas d'ID. Conséquemment, pour faire le lien entre les individus du format « .sample » et ceux du format « .gen » il faut utiliser la colonne du fichier « .gen » et la ligne du « .sample ». Par exemple, les 3 premières colonnes du format « .gen » vont appartenir à l'individu dans la 1re ligne du format « .sample » associé; les 3 deuxièmes colonnes du format « .gen » vont appartenir à l'individu sur la 2e du format « .sample »associé. De cette manière il est possible d'obtenir l'ID du patient.

Les 5 premières colonnes de chaque ligne contiennent : le SNP_ID (ou chromosome), le RS_ID (ou ID de la variation), la position dans la chaine d'ADN, l'allèle codé A et l'allèle codé B. Les 3 prochaines colonnes représentent la probabilité d'occurrence des trois génotypes, AA, AB et BB. Ces probabilités doivent totaliser 1 (voir la figure 2.2).

SNP_ID	RS_ID	Position	Allele A	Allele B	Probabilités individu 1			Probabilités individu 2						Probabilités individu N		
1	rs120487	1491251	T	C	0.076	0.924	0	1	0	0	1	0	0	1	0	0
1	rs349628	1492875	CAT	C	0.009	0.989	0.002	0.997	0.003	0	1	0	0	1	0	0
1	rs880051	1493727	G	A	0.085	0.915	0	1	0	0	1	0	0	1	0	0
1	rs383546	1494106	C	CA	0.009	0.991	0	1	0	0	1	0	0	1	0	0
1	rs666734	1495083	G	C	0.06	0.94	0	1	0	0	1	0	0	1	0	0
1	rs353842	1495130	C	CAG	0	1	0	1	0	0	1	0	0	1	0	0
1	rs124108	1496145	T	C	0.068	0.932	0	1	0	0	1	0	0	1	0	0
1	rs376617	1497008	T	C	0	1	0	1	0	0	1	0	0	1	0	0
1	rs376616	1497201	A	C	0.009	0.991	0	1	0	0	1	0	0	1	0	0
1	rs358858	1497641	TTTTC	T	0.068	0.932	0	1	0	0	1	0	0	0.994	0.006	0
1	rs943946	1499298	A	G	0	1	0	1	0	0	1	0	0	1	0	0

Figure 2-2 Le fichier de format « .gen »

2.3 Le volume de l'ensemble de données

Chaque ensemble de données est composé par un fichier « .sample » et plusieurs fichiers « .gen ». Le jeu de données utilisé pour ce projet comprend un fichier « .sample » contenant 3409 individus et 71 fichiers « .gen ». Le fichier « .sample » est relativement petit, incluant 3411 lignes et 7 colonnes, d'une taille de 70 kilooctets (KO). Par contre, les « .gen » sont d'un volume considérable. Chaque fichier « .gen » a approximativement 84500 lignes et 10232 colonnes (c.-à-d. 3 colonnes par individu plus 5 de données partagées), d'une taille de presque 2 gigaoctets (GO) par fichier. La taille de chaque fichier « .gen » rend impossible son édition par un éditeur de texte standard (par exemple UltraEdit ou Notepad++) ou par un tableur (par exemple Microsoft Excel). Ceci est une caractéristique du domaine du Big Data ou les données sont trop volumineuses pour être éditées à l'aide d'outils classiques.

Donc, pour ce projet appliqué, il s'agit de traiter un ensemble de données, en format texte, d'approximativement 142 GO. L'ensemble de fichiers reçus est compressé afin d'optimiser l'utilisation de l'espace disque et d'en faciliter le transfert, ce qui réduit la taille à 13 GO.

2.4 Conclusion

Ce chapitre a présenté le format et contenu des fichiers « .gen » et « .sample » ainsi que la difficulté des outils bureautiques pour gérer des fichiers de volume considérable. La complexité ne se trouve seulement dans la taille des fichiers à traiter, mais aussi dans le détail du contenu de chaque fichier.

Le chapitre suivant présente une solution à la gestion des fichiers d'Oxford et les défis à faire face lors de sa conception.

CHAPITRE 3

LA CONCEPTION D'UNE SOLUTION

Ce chapitre présente les défis de la solution, la structure et fonctionnement des nouveaux fichiers générés à l'issue du traitement des fichiers « .gen » et « .sample » et les étapes pour y arriver à les créer.

3.1 Les défis

1. Faire le lien entre les individus du fichier « .gen » et ceux du fichier « .sample », puisqu'il n'y a pas d'ID dans les fichiers « .gen »;
2. Être capable de tracer facilement les données source avec leur destination, étant donné que c'est difficile à se retrouver dans les fichiers « .gen », que leur volume considérable rende impossible ça manipulation par des outils standards et de qu'il faut avoir certaines connaissances des commandes Shell Unix pour les ouvrir;
3. Intégrer les génotypes imputés d'IMPUTE2 aux données d'ADVANCE, étant donné qu'il faut conserver les données de l'individu, les visites et les phénotypes d'ADVANCE, mais avec les génotypes des « .gen ».
4. Utiliser au maximum les ressources computationnelles disponibles, parce qu'il faut optimiser l'utilisation la mémoire RAM disponible et les cœurs du processeur.

3.2 Vue d'ensemble de la transformation des fichiers OXFORD

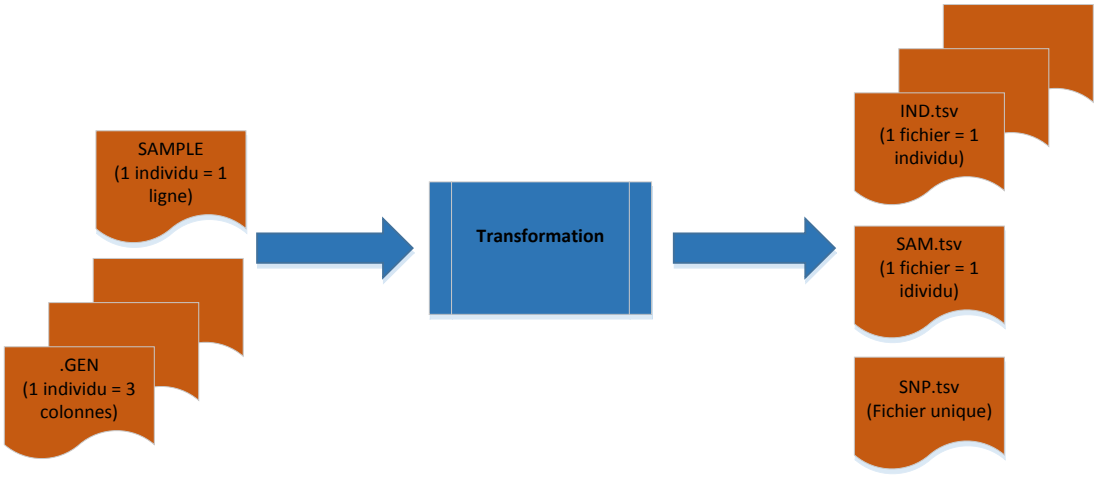


Figure 3-1 Vue d'ensemble de la transformation des fichiers OXFORD

Les principales étapes effectuées pour arriver à générer les fichiers « IND », « SAM » et « SNP », décrits dans les sous-chapitres 3.3 à 3.5, sont les suivantes :

1. Assurer l'existence du fichier « .sample », qui conduira l'exécution du script de transformation *gentotsv.py* (voir ANNEXE I, script gentotsv.py).
2. Parcourir le fichier « .sample » et générer les fichiers « SAM » et « IND » de tous les individus. Dans le cas des « IND », je génère des fichiers vides que je mets à jour à une étape postérieur. Le nom de chaque fichier et sauvegardé dans une liste de fichiers générés.
3. Créer le fichier « SNP ». Je visite chaque fichier « .gen » et je récupère les 5 premières colonnes de chaque ligne, car c'est la partie du fichier partagé par tous les individus de chaque ligne.
4. Mettre à jour les fichiers « IND ». Parcourir chaque fichier « .gen », récupérer les probabilités des génotypes de chaque individu et mettre à jour le fichier « IND » correspondant qui se trouve dans la liste des fichiers génère.
5. Les étapes 3 et 4 sont exécutées en parallèle.

3.3 « .sample » à « .tsv ». Fichiers « SAM »

Dans le but de simplifier la traçabilité et de ne pas avoir à faire la lecture complète du fichier « .sample » lors des étapes subséquentes du cycle de conversion, j'ai fractionné le fichier en plusieurs fichiers. Chaque fichier représente un individu. Chaque fichier est nommé d'une manière qui permet de faire le lien avec les individus des fichiers « .gen ». Ex.

SAM_4_10522_10522.tsv

SAM	4	10522	10522	tsv
Type fichier. Valeur fixe	Ligne où se trouve l'individu dans le fichier	#ID 1 de l'individu	#ID 2 de l'individu	Extension du fichier. Valeur fixe

Figure 3-2 Nomenclature du fichier « SAM »

En outre, les colonnes ont été converties en lignes pour simplifier sa lecture.

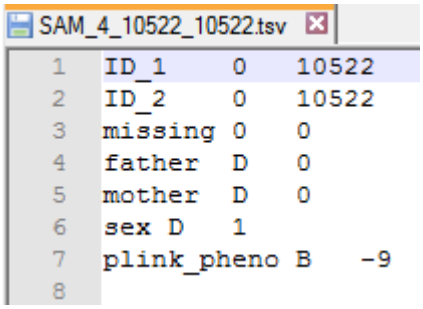


Figure 3-3 Structure du fichier « SAM »

À la fin de la transformation, 3409 fichiers « SAM » sont obtenus, soit le nombre d'individus du jeu de données initiales.

3.4 « .gen » à « .tsv ». Fichiers « IND »

Afin de simplifier la traçabilité et ne pas avoir à faire la lecture complète tous les fichiers « .gen » pour chaque individu dans les étapes suivantes du cycle de conversion, chaque fichier est fractionné en plusieurs fichiers. Chaque fichier représente un individu et il contient

l'ensemble de toutes les probabilités des variations de tous les fichiers « .gen ». C'est-à-dire, chaque nouveau fichier « IND » (voir figure 3.5) inclus la totalité des probabilités de tout le jeu de données. Chaque fichier est nommé d'une manière qui permet de faire le lien avec les individus du fichier « .sample ». Ex.

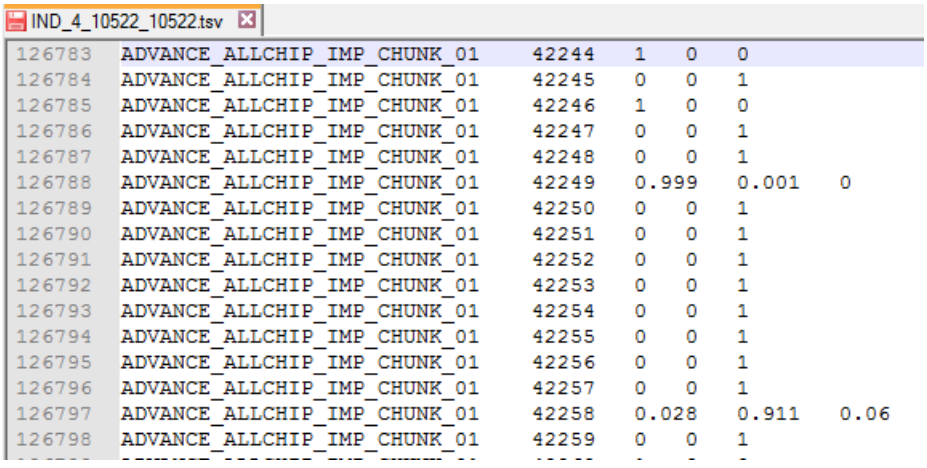
IND_4_10522_10522.tsv

IND	4	10522	10522	tsv
Type fichier. Valeur fixe	Colonne où se trouve l'individu dans le fichier	#ID 1 de l'individu (du fichier « .sample »)	#ID 2 de l'individu (du fichier « .sample »)	Extension du fichier. Valeur fixe

Figure 3-4 Nomenclature du fichier « IND »

Le lien entre les individus du fichier « .sample » et « .gen » se fait par ligne et colonne. Tel que décrit à l'étape 2.2, les colonnes des individus dans le fichier « .gen » gardent le même ordre que les lignes du fichier « .sample ». Conséquemment les trois 1res colonnes de probabilités du fichier « .gen » appartiennent à l'individu de la 1re ligne du fichier « .sample » ; les trois 2^{e} colonnes de probabilités du fichier « .gen » appartiennent à l'individu de la 2^{e} ligne du « .sample » ; et ainsi de suite.

En outre, chaque fichier <IND> doit avoir son fichier « .gen », la source et la ligne des probabilités. Ces deux valeurs sont utilisées à titre de clé primaire et permettent ainsi de faire le lien avec le troisième fichier qui doit être créé, le fichier contenant les « SNP ».


IND_4_10522_10522.tsv

126783	ADVANCE_ALLCHIP_IMP_CHUNK_01	42244	1	0	0	
126784	ADVANCE_ALLCHIP_IMP_CHUNK_01	42245	0	0	1	
126785	ADVANCE_ALLCHIP_IMP_CHUNK_01	42246	1	0	0	
126786	ADVANCE_ALLCHIP_IMP_CHUNK_01	42247	0	0	1	
126787	ADVANCE_ALLCHIP_IMP_CHUNK_01	42248	0	0	1	
126788	ADVANCE_ALLCHIP_IMP_CHUNK_01	42249	0.999	0.001	0	
126789	ADVANCE_ALLCHIP_IMP_CHUNK_01	42250	0	0	1	
126790	ADVANCE_ALLCHIP_IMP_CHUNK_01	42251	0	0	1	
126791	ADVANCE_ALLCHIP_IMP_CHUNK_01	42252	0	0	1	
126792	ADVANCE_ALLCHIP_IMP_CHUNK_01	42253	0	0	1	
126793	ADVANCE_ALLCHIP_IMP_CHUNK_01	42254	0	0	1	
126794	ADVANCE_ALLCHIP_IMP_CHUNK_01	42255	0	0	1	
126795	ADVANCE_ALLCHIP_IMP_CHUNK_01	42256	0	0	1	
126796	ADVANCE_ALLCHIP_IMP_CHUNK_01	42257	0	0	1	
126797	ADVANCE_ALLCHIP_IMP_CHUNK_01	42258	0.028	0.911	0.06	
126798	ADVANCE_ALLCHIP_IMP_CHUNK_01	42259	0	0	1	

Figure 3-5 Structure du fichier « IND »

À la fin de la transformation, 3409 fichiers de format « IND » sont générés, soit le nombre d'individus du jeu de données initial.

3.5 « .gen » à « .tsv ». Fichier « SNP »

Un troisième fichier doit être généré, c'est le fichier « SNP » (voir figure 3.6). Ce fichier contient les 5 premières colonnes de chaque fichier « .gen ». C'est un seul fichier avec la partie partagée entre les individus du « .gen ». Ce fichier contient les chromosomes, les variantes, la position et les allèles du jeu de données complet. Similaire au cas des fichiers « IND », il faut ajouter, au fichier, le nom du fichier source et sa position dans le fichier. Ces deux colonnes agissent à titre de clé primaire pour faire le lien avec les fichiers « IND ».

SNP.tsv

126783	ADVANCE_ALLCHIP_IMP_CHUNK_01	42244	1	rs11582043	69883108	T	C
126784	ADVANCE_ALLCHIP_IMP_CHUNK_01	42245	1	rs7553552	69883201	G	T
126785	ADVANCE_ALLCHIP_IMP_CHUNK_01	42246	1	rs10889821	69883240	A	T
126786	ADVANCE_ALLCHIP_IMP_CHUNK_01	42247	1	rs35812486	69883464	AT	A
126787	ADVANCE_ALLCHIP_IMP_CHUNK_01	42248	1	rs12074578	69883953	T	A
126788	ADVANCE_ALLCHIP_IMP_CHUNK_01	42249	1	rs147558318	69884335	G	A
126789	ADVANCE_ALLCHIP_IMP_CHUNK_01	42250	1	rs57122951	69884853	C	CA
126790	ADVANCE_ALLCHIP_IMP_CHUNK_01	42251	1	rs4529654	69884884	A	G
126791	ADVANCE_ALLCHIP_IMP_CHUNK_01	42252	1	rs60403061	69885159	CT	C
126792	ADVANCE_ALLCHIP_IMP_CHUNK_01	42253	1	rs868402	69885329	G	T
126793	ADVANCE_ALLCHIP_IMP_CHUNK_01	42254	1	rs1887044	69885376	C	T
126794	ADVANCE_ALLCHIP_IMP_CHUNK_01	42255	1	rs1926257	69886629	A	G
126795	ADVANCE_ALLCHIP_IMP_CHUNK_01	42256	1	rs4388647	69886702	T	C
126796	ADVANCE_ALLCHIP_IMP_CHUNK_01	42257	1	rs1926258	69886839	G	A

Figure 3-6 Structure du fichier « SNP »

3.6 Conclusion

Cette étape de transformation est une étape incontournable du projet de migration des formats. Certaines décisions de conception qui ont été soulevées lors de cette étape ont été : 1) à quel moment faire la transformation (c.-à-d. au tout début du traitement ou lors de l'intégration avec les données d'ADVANCE); et 2) avec quel critère fractionner les fichiers.

La décision a été prise de transformer des fichiers « .gen » au format « .tsv » au tout début du traitement, à l'aide d'un nouveau script, de manière à limiter les responsabilités de chaque script et ne pas avoir à modifier les scripts existants. L'option de fractionner les fichiers par individu pour contribuer à la traçabilité des données a été prise.

Le chapitre suivant présente la préparation de l'environnement de travail avec les outils nécessaires pour répondre le projet de Simon Grondin et l'étendre avec les génotypes imputés.

CHAPITRE 4

LA PRÉPARATION DE L'ENVIRONNEMENT DE TRAVAIL

Ce chapitre présente une synthèse des outils qu'il a fallu maitriser ainsi que la préparation de l'environnement pour effectuer cette conversion.

4.1 Le défi d'apprentissage du domaine du Big Data

Le défi principal, que j'ai dû affronter, est l'apprentissage d'un grand nombre de technologies utilisées dans une solution moderne de Big Data. N'ayant que de l'expérience sur des bases données relationnelles (par exemple : Oracle, MSSQL, Sybase) opérant sur le système d'opération Microsoft Windows, il a été nécessaire d'acquérir des connaissances sur les technologies du logiciel libre, les machines virtuelles, le logiciel de gestion des versions GitHub, le système d'opération Linux, le langage de programmation Python, la base de données libre PostgreSQL, le logiciel libre de compilation SBT, le langage de programmation Scala, le format de fichier Big Data Apache Parquet, la plateforme Big Data Apache Spark et le logiciel de sérialisation Apache Avro. Ceci a demandé un effort considérable de recherche, des discussions avec les autres membres de l'équipe et des nombreux essais et erreurs.

4.2 Les technologies utilisées

4.2.1 Oracle VM VirtualBox

VirtualBox [13][14]est un logiciel libre de virtualisation conçu par Oracle où il y a une machine hôte avec un système hôte qu'hébergent plusieurs machines virtuelles (machine invitée) avec des systèmes d'exploitation invités.

Compte tenu de que j'utilise un *laptop* avec le système d'exploitation Windows, j'ai dû me créer une machine virtuelle Linux afin de pouvoir reproduire le processus de migration d'ADVANCE et dbSNP vers PostgreSQL, adapter les fichiers « .gen » vers le format ADAM et générer les fichiers Parquet pour qu'ils puissent être utilisés par la technologie Big Data Spark. La version utilisée est la 5.0 de VirtualBox a été utilisée pour ce projet.

4.2.2 Linux Ubuntu

Ubuntu [15][16]est un système d'exploitation *libre* basé sur la distribution Linux Debian. C'est le système d'exploitation invité (« *guest* ») utilise dans la machine virtuelle de ce projet. Son installation est simple et intuitive grâce à son interface graphique. La complexité se situe dans l'utilisation de lignes de commandes qui servent pour installer et configurer les outils de développement nécessaires au projet, la configuration et le partage de ressources avec le système d'exploitation hôte (*host* Windows). La version utilisée d'Ubuntu pour ce projet est la 15.10.

4.2.3 Python

Le langage de programmation Python [17][18] est très utilisé dans le domaine de la génomique. C'est un langage de programmation libre, orienté objet, multi paradigme et multiplateforme. Les bio-informaticiens du CRCHUM utilisent ce langage de programmation actuellement.

Chaque fois qu'il est nécessaire d'apprendre un nouveau langage programmation, il faut se familiariser avec sa syntaxe, ses commandes, ses avantages et contraintes. Les scripts pour l'importation des bases de données dbSNP et ADVANCE vers la base de données PostgreSQL ainsi que la génération des scripts pré-ADAM (les fichiers .tsv générés par le script *export.py* qui seront ensuite utilisés pour générer les fichiers Parquet-ADAM. Voir figure 5.4) étaient déjà développés avec ce langage donc il a été nécessaire de maitrise le langage afin de développer de nouveaux scripts et adapter ceux qui étaient déjà présents au début du projet. La version utilisée 3.5 de Python a été utilisée pour ce projet.

4.2.4 PostgreSQL

PostgreSQL [19][20] est un système de gestion de base de données libre, relationnelle et orientée objet. Au début de ce projet, la base de données ADVANCE était en format PostgreSQL. Il a donc été nécessaire d'importer le fichier dépôt (*dump*) afin de pouvoir intégrer les visites et phénotypes des individus aux génotypes imputés d'IMPUTE2. Il a aussi été nécessaire de créer le schéma « *prognomix* » (schéma propriétaire des données utilisées au CRCHUM) afin de pouvoir effectuer l'importation de données. En outre, il a été nécessaire d'importer dans PostgreSQL les données de dbSNP, car elles sont utilisées lors de la création des fichiers pré-ADAM. La version de PostgreSQL utilisée pour ce projet est la 9.4.

4.2.5 SBT

Le logiciel libre SBT [21][22] est un moteur de compilation libre pour les projets qui utilisent le langage de programmation Scala et Java. Sa fonction principale est d'automatiser la création des projets. Dans ce projet, SBT est utilisée dans la création des fichiers Parquet, car la logique pour la transformation des fichiers pré-ADAM en format ADAM a été développée avec le langage de programmation Scala et utilise des objets Java. La difficulté principale de l'utilisation de SBT est dans sa configuration pour l'allocation de la mémoire tas (de l'anglais « *heap* »). Les valeurs par défaut offertes par SBT n'étaient pas suffisantes pour supporter la compilation du volume de données des fichiers pré-ADAM, et a généré, à plusieurs reprises, des débordements de mémoire. La version utilisée de SBT pour ce projet est la version 0.13.

4.2.6 Scala

Le langage de programmation Scala [23][24] est un langage de programmation orienté objet et fonctionnel qui supporte la syntaxe Scala et Java. Scala est le langage de programmation par défaut du cadriciel Apache Spark pour le Big Data et par conséquent, et le plus efficace à utiliser dans des projets Big Data qui veulent utiliser toute la puissance de Spark.

Pour ce projet, il a été nécessaire de modifier le script Scala existant afin de pouvoir ajouter au cadriciel ADAM les nouveaux champs de données en provenance des fichiers « .gen » /» .sample » (voir ANNEXE « Cartografie ADAM ». Pour ce projet, la version utilisée de Scala est la version 2.12.

4.2.7 Apache Parquet

Parquet [25][26] est un format de stockage, en colonnes, avec un haut niveau de compression qui est très efficace pour le traitement de données massives. Il est supporté par le langage Spark SQL ce qui le rend très populaire actuellement dans le domaine du Big Data. Le cadriciel ADAM tire avantage des fichiers en format Parquet.

Les fichiers Parquet sont générés par le script Scala de conversion des fichiers pré-ADAM vers ADAM. Les données dans ces fichiers sont représentées par un schéma de hiérarchies (voir annexe « Schéma PARQUET »). La version d'Apache Parquet utilisée dans ce projet est la version 1.6.

4.2.8 Apache Spark

Spark [27][28] est un cadriciel libre pour le traitement de données massives sur une grappe d'ordinateurs qui permet l'élasticité automatique. Il fournit aux programmeurs une interface de programmation d'applications centrée sur les ensembles de données distribuées résilientes (c.-à-d. *Resilient Distributed Datastore (RDD)*) qui permet une vitesse de traitement de l'information jusqu'à cent fois plus rapide que par l'utilisation du langage de programmation MapReduce.

Il a été nécessaire de maitrise la technologie Spark SQL afin de pouvoir tester que les données sources ont été converties adéquatement vers le format ADAM. La difficulté initiale de l'utilisation de Spark se situe au niveau de la configuration pour l'allocation de mémoire tas. Les valeurs par défaut ne sont pas suffisantes pour traiter en mémoire un volume

considérable de données, en générant à plusieurs reprises des débordements de mémoire. La version de Spark utilisée pour ce projet est la version 2.11.

4.2.9 Apache Avro

Avro [29][30] est un système de sérialisation libre de données massives pour le Big Data. Les schémas d'ADAM sont implémentés avec l'aide de Avro.

Pour ce projet, il a été nécessaire d'apprendre comment ajouter certains champs manquants dans le schéma original reçu au début du projet et comment régénérer les objets du schéma. La version utilisée de Avro dans ce projet est la version 1.7.7.

4.2.10 GitHub

GitHub [31][32] un service web d'hébergement et de gestion de logiciels. Les sources du projet sont hébergées dans des différents entrepôts de GitHub. Dans ce projet, il a été nécessaire d'apprendre le fonctionnement des *branches* et d*es « pull requests ».*

4.3 Conclusion

Le Big Data n'est pas composé d'une seule technologie, mais plutôt d'un ensemble de technologies qui s'intègrent et contribuent à l'objectif de traiter un volume massif de données efficacement. Ce chapitre a présenté l'ensemble des technologies qui ont dû être maitrisées afin d'effectuer ce projet de recherche appliquée.

Le chapitre suivant présente la solution et des stratégies que j'ai appliquées pour optimiser l'exécution du script de transformation de fichiers d'Oxford en « .tsv ».

CHAPITRE 5

LA CONCEPTION ET LA RÉALISATION

Ce chapitre décrit la conception de la solution afin d'optimiser l'utilisation de ressources mises à disposition pour les preuves de concepts. On y présente aussi le diagramme de la solution proposée et implémentée.

5.1 Le traitement des fichiers en parallèle

Faire le traitement efficace et rapide des fichiers de données massives, par exemple les fichiers « .gen » requièrent beaucoup de temps de traitement ainsi que beaucoup de ressources informatiques. La situation devenait difficile pour les bioinformaticiens du CRCHUM de traiter ces données avec des technologies classiques de bases de données relationnelles. Afin d'optimiser l'utilisation de processeurs multicœurs, la programmation, le traitement et la création du fichier « SNP.tsv » et des fichiers « IND.tsv » ont été effectués de manière à ce qu'ils puissent s'exécuter de manière indépendante (c.-à-d. en parallèle).

La conception du script *gentotsv.py* accepte, à titre de paramètre d'entrée, le nombre de processus (de l'anglais « *threads »*) à exécuter en parallèle. Par défaut, le nombre processus correspond au nombre de cœurs des processeurs disponibles. Par exemple, si il y a un processeur quatre-cœurs, tout au début de la transformation (à l'instant t1 de la figure 5.1) un cœur sera utilisé pour la création du fichier « SNP » et les autres trois seront utilisés pour la création des fichiers « IND ». Une fois terminée la création du fichier « SNP », par exemple à l'instant t2 de la figure 5.1, la totalité des cœurs sera utilisée pour la création des fichiers « IND ».

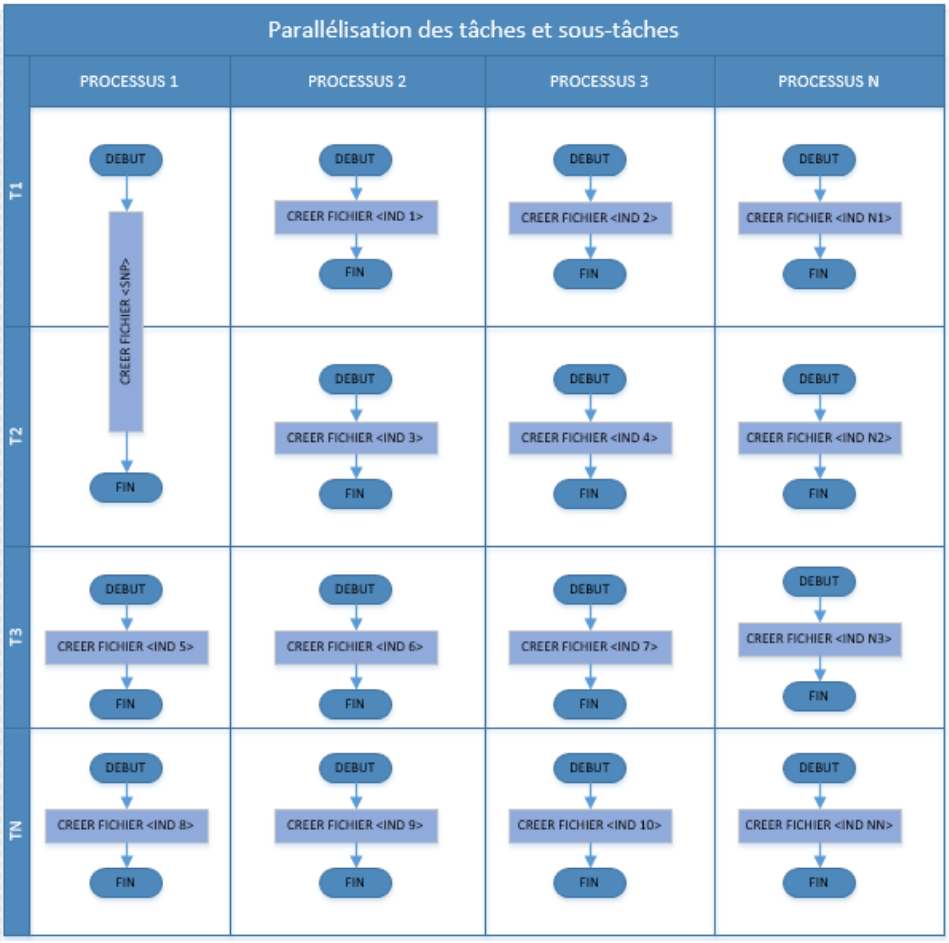


Figure 5-1 Exécution en parallèle

5.2 La gestion de la mémoire, des entrées-sorties et des processeurs

Une autre décision de conception qui est requise est d'évaluer le compromis le plus efficace entre l'utilisation de la mémoire vive et les lectures/écritures au disque dur.

Le langage de programmation Python propose de faire la lecture d'un fichier de texte à l'aide de deux options, soit : 1) ligne par ligne; ou 2) en le chargeant au complet en mémoire. La section suivante discute de ces options.

5.2.1 Lecture du fichier ligne par ligne

Avantages :

- Utilisation uniforme du processeur;
- Faible utilisation de mémoire vive;
- Plus de threads d'exécution, car il n'y a pas de risque de débordement de mémoire.

Inconvénients :

- Très grand nombre de lectures et écritures sur disque;
- Le processeur n'est pas utilisé à 100%, car il est limité par la vitesse du disque.

Cette option a été utilisée et étudiée. La figure 5.2 décrit qu'après 10 minutes d'exécution l'utilisation de la mémoire RAM est demeurée constante, oscillant entre 650-670 mégaoctets, et les quatre-cœurs ont eu une utilisation constante dans l'ordre du 75%.

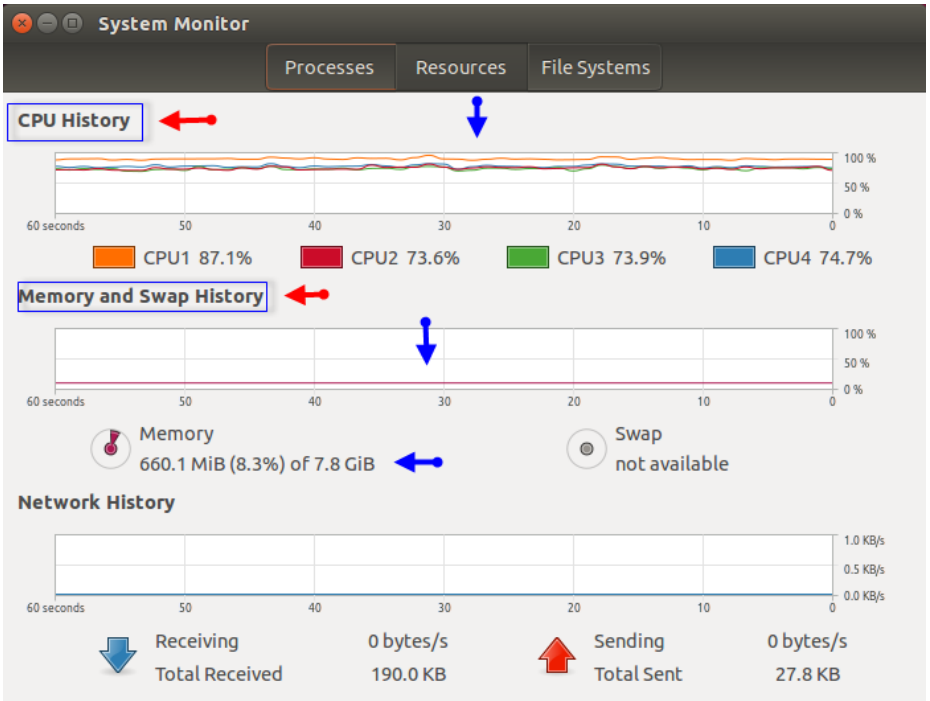


Figure 5-2 Exécution avec lecture ligne par ligne

5.2.2 Chargement du fichier au complet en mémoire

Avantages

- Très peu des entrées/sorties sur disque, car chaque fichier est chargé tout d'un coup en mémoire;
- Utilisation du processeur à 100% au moment de la lecture du fichier, car il se trouve en mémoire.

Inconvénients

- Utilisation non uniforme du processeur.
- Considérable utilisation de mémoire vive, car il dépend du volume du fichier à traiter.
- La vitesse de chargement en mémoire dépend de la vitesse du disque dur.
- Le nombre de « *threads »* d'exécution est limité par la quantité de mémoire disponible.
- Besoin d'ajouter des contrôles additionnels afin de ne pas avoir des débordements de mémoire.

Cette option a aussi été étudiée. La figure 5.3 décrit qu'après 10 minutes d'exécution l'utilisation de la mémoire RAM monte et descende considérablement en raison du chargement et libération de mémoire, oscillante entre 5 – 7.5 gigaoctets, et les quatre-cœurs ont eu une utilisation qui varie entre 50% à 100%.

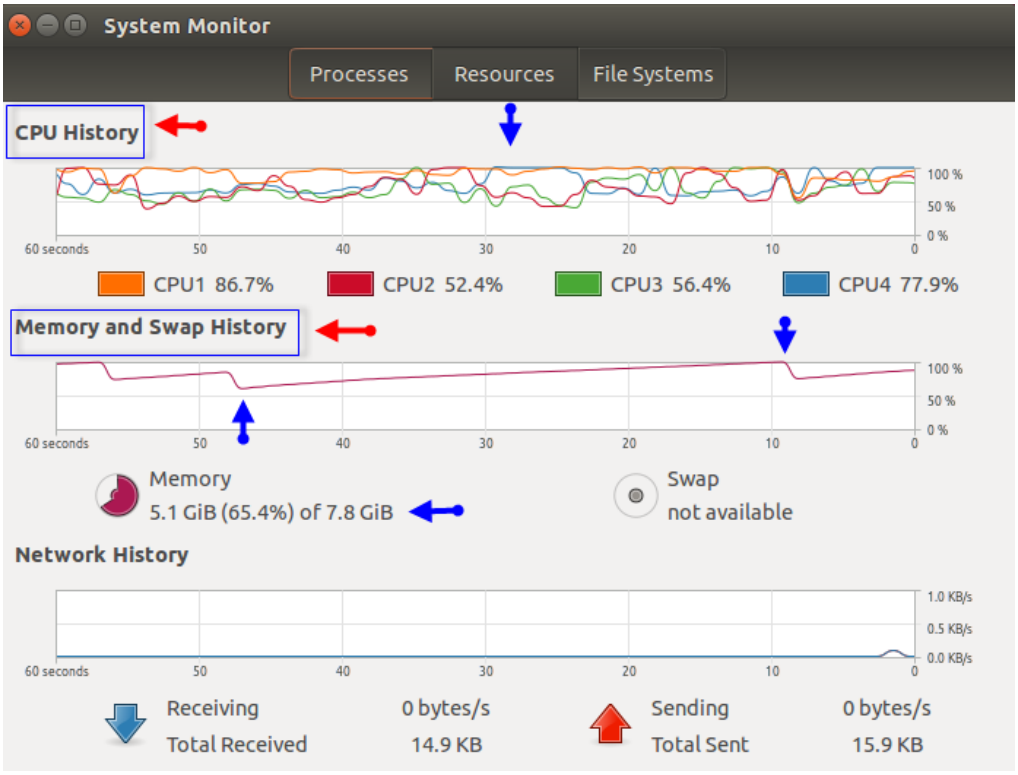


Figure 5-3 Exécution avec chargement au complet en mémoire

Suite à ces expériences, la solution optimale fait la lecture des fichiers ligne par ligne, car suite à l'exécution des deux alternatives, pour des périodes de plus de 24 heures, il n'a pas été possible de noter des différences notables. Conséquemment, la lecture ligne par ligne évitant des débordements de mémoire a été privilégiée.

Avant de démarrer le développement d'une solution finale, il est nécessaire de connaitre les caractéristiques du serveur où la solution va s'exécuter. Ceci permet de pouvoir identifier les *bottleneck* de la solution et analyser les différentes possibilités d'optimisation du langage de programmation. Dans le cas de mon projet, il a été remarqué que le goulot d'étranglement se trouve au niveau de la vitesse du disque. L'utilisation de la technologie SDD et la disponibilité de plus de mémoire vive aurais probablement permis de charger le fichier au complet en mémoire et obtenir un gain significatif en performance d'exécution.

5.3 Vue d'ensemble de la solution

La figure 5.4 présente la solution bout en bout du projet de Simon Grondin et mon projet, le processus complet de transformation des sources de données en fichiers Parquet-ADAM.

Voici la description des étapes :

1. La première étape est la transformation des sources de données, en format d'origine, à un format plus simple à manipuler (étapes 1a, 1b, 1c). Les sources de données sont indépendantes entre elles donc les conversions peuvent s'exécuter dans n'importer quel ordre c'est pour cette raison que je leurs ait donné la même préséance. L'étape « 1a » convertit les fichiers « .gen » et « .sample » en « .tsv » (déjà discuté dans le chapitre 3). L'étape « 1b » convertit/importe le fichier « *dump »* de ADVANCE en format PostgreSQL.). L'étape « 1c » convertit le fichier « .vcf » avec dbSNP en fichier intermédiaire « .tsv »
2. L'étape 2 convertit/importe le fichier intermédiaire de dbSNP en format PostgreSQL.
3. L'étape 3 fait la fusion des toutes les nouvelles sources de données (schéma PostgreSQL avec ADVANCE et dbSNP et fichiers « SNP.tsv », « SAM.tsv » et « IND.tsv ») et générés les « .tsv » pré-ADAM.
4. L'étape 4 est optionnelle. S'il faut étendre ADAM avec de nouveaux champs, il faut régénérer le schéma Avro.
5. Finalement, l'étape 5 est de transformer les fichiers « .tsv » pré-ADAM en fichiers Parquet – ADAM qui seront lus avec Spark.

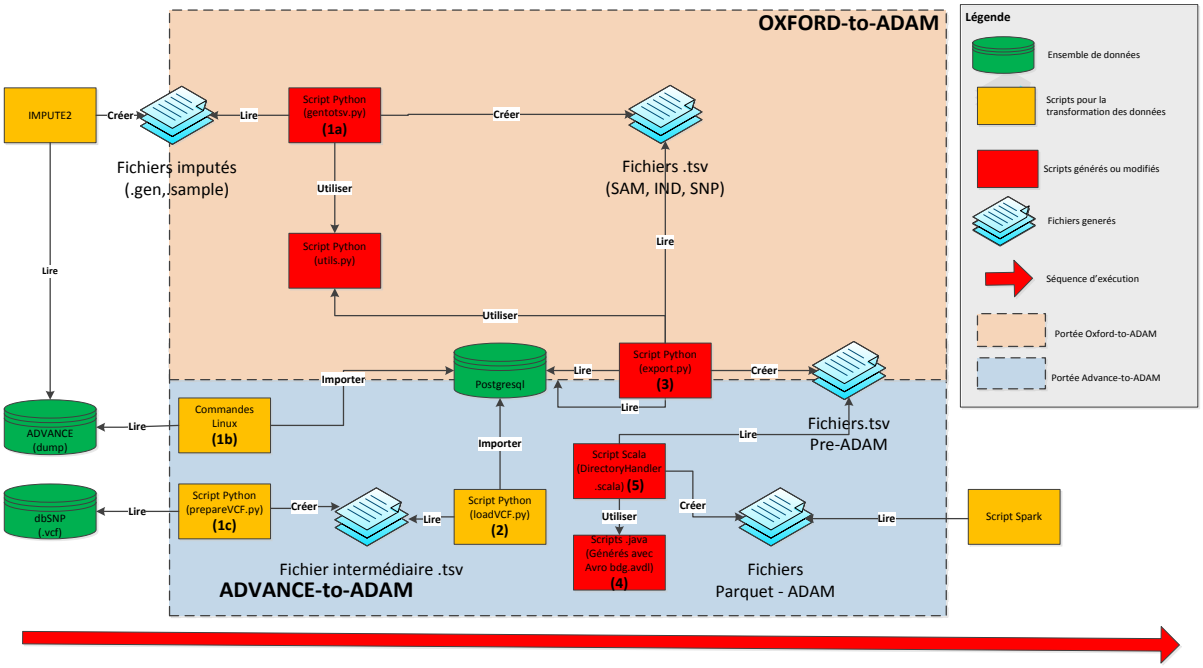


Figure 5-4 Vue d'ensemble de la solution

5.4 Conclusion

Ce chapitre a présenté deux caractéristiques du script du script *gentotsv.py*, qui s'occupe de la transformation des fichiers d'Oxford en « .tsv », l'exécution des threads et la gestion des ressources. Finalement je présente la solution intégrale, avec les parties que j'ai créées et modifiée du projet de Simon Grondin, pour la générer un nouveau jeu de données contenant les génotypes imputés d'IMPUTE2, les données cliniques des patients d'ADVANCE et les variations génétiques de dbSNP.

Le chapitre suivant présente certaines alternatives de solution à explorer, ma perception de cette expérience et des recommandations pour des travaux futures.

CHAPITRE 6

DISCUSSION

Ce chapitre présente des aspects que j'aurai voulu explorer et points positifs et négatifs que j'ai retenus de cette expérience ainsi que certaines recommandations pour la suite du projet.

6.1 Points d'exploration

6.1.1 La convivialité d'Apache Spark avec ses voisins

Spark est particulièrement gourmande en utilisation de mémoire donc il faut analyser et tester comment il se comporte avec d'autres logiciels qui partagent le même serveur ou grappe de serveurs (*server cluster*). Une pénurie de ressources computationnelles lors de l'exécution de Spark peut interférer avec d'autres tâches qui essaient de s'exécuter au même moment avec les mêmes ressources.

6.1.2 L'expérimentation avec Apache Flink

À date, et depuis les sources de données que nous avons à notre disposition, faire l'analyse de données à partir des lots semble d'être la seule alternative. Mais on ne sait pas si au fur et à mesure que nous allons incorporer des sources de données au format étendu d'ADAM nous allons nous trouver avec un scénario au il vaut mieux faire des analyses de données en temps réel. Spark offre déjà la fonctionnalité de « *micro-batching* » pour faire face à ce type d'exigence par contre, Apache Flink [34][35], un nouveau cadriciel libre pour l'analyse de données massives, a été déjà pensé dès sa conception pour répondre à ce type de besoin. Faire un *« benchmarking »* des deux technologies dans différents scénarios peut faire l'objet d'un nouveau projet.

6.1.3 Changement du moment de traitement des fichiers « .gen » et « .sample »

Présentement dans ma solution j'ai opté pour transformer d'abord les fichiers d'Oxford au format « .tsv », avec le script « *gentotsv.py* », et ensuite faire la fusion avec les données d'ADVANCE et dbSNP avec le script « *export.py* ». Chaque script a une seule responsabilité. Une alternative à expérimenter est de faire les deux étapes, transformation et fusion, à la volée (de l'anglais *on-the-fly*) au moment de la préparation des fichiers pré-ADAM. Cette approche accrue la complexité et responsabilités du script « *export.py* » mais probablement permet générer des améliorations dans le temps total requis pour l'exécution de toutes les étapes des projets A2A et O2A (voir figure 5.4) et aussi dans l'utilisation des ressources, principalement dans l'utilisation de l'espace du disque.

6.2 Points positifs

6.2.1 La participation d'un expert dans le domaine d'affaires

La participation d'un expert dans le domaine d'affaire, comme c'est le cas de Beatriz, capable d'expliquer le jargon utilisé dans un domaine si complexe comme la génomique et appartenant à l'équipe QnGene CRCHUM a été grandement appréciée.

6.2.2 L'autonomie

Le fait de pouvoir gérer mes horaires et ne pas avoir eu à suivre un plan de travail rigoureux m'a permis trouver une balance entre les exigences du projet et ma réalité personnelle et familiale.

6.2.3 L'apprentissage de nouvelles technologies

Tel que mentionné au Chapitre 4, l'apprentissage de nombreuses technologies a été nécessaire.

6.3 Points négatifs

6.3.1 La distribution de la documentation

La documentation du projet QnGene CRCHUM n'est pas centralisée ce qui rend difficile de se retrouver dans le projet. Certains documents sont dans GoogleDocs, d'autre dans GitHub, d'autres simplement sur l'ordinateur de travail des participants.

6.3.2 La complexité dans la compréhension du domaine

L'information disponible sur l'internet, concernant la terminologie utilisée dans le domaine de la génomique est complexe. Je remarque qu'il y a une difficulté généralisée des bio-informaticiens, biologistes et médecins, pour exprimer de manière simple des définitions complexes.

6.3.3 La exposition élevée du domaine aux changements de contexte

Le domaine de la recherche en santé, pour sa grande dépendance à l'appui des parties intéressées (*stakeholders*), est un domaine où il faut s'attendre à beaucoup de changements et d'ajustements tout au long du projet.

6.4 Recommandations pour des travaux futurs

1. La documentation complète de toutes les parties du projet QnGene CRCHUM et de toutes les intervenantes devrait se retrouver dans un outil centralisé de travail collaboratif du genre Sharepoint, avec l'option de recherche par mot-clé.
2. Tout le processus de transformation d'ADVANCE-dbSNP-OXFORD en ADAM devrait être automatisé pour ne pas avoir à exécuter un script à la fois.

3. Les sources de données devraient se trouver dans un serveur de l'ÉTS pour que tous les participants du projet puissent accéder. Présentement chaque membre de l'équipe géré ses propres sources.
4. Un logiciel de gestion de versions, du style Tortoise SVN, commun pour toute l'équipe QnGene, devrait être mis en place pour le partage et suivi des scripts entre les membres.

CONCLUSION

Ce projet consistait en l'adaptation du format ADAM pour y ajouter les génotypes imputés provenant des fichiers générés par IMPUTE2 ainsi que son intégration avec les données cliniques provenant d'ADVANCE et dbSNP. En outre j'ai dû reproduire et documenter les étapes requises pour importer ADVANCE et dbSNP à la base de données PostgreSQL, car la documentation existante était de très haut niveau.

En cour de route j'ai appris à utiliser un grand nombre de technologies utilisées pour le Big Data, à gérer de très gros fichiers, à tester des concepts de programmation avancés (par exemple l'exécution en parallèle et concurrente ainsi que le suivi de l'utilisation des ressources pour optimiser le traitement),. De plus j'ai fait l'apprentissage de concepts et de la terminologie de la génomique qui est un domaine d'affaires nouveau pour moi.

Comme derniers mots, j'aimerais soulever l'importance de projets effectués en collaboration, par exemple, celui-ci a été réalisé en collaboration avec le Centre de Recherche de l'Université de Montréal et l'École de Technologie Supérieure. Ce type de projet offre l'opportunité aux étudiants de mettre en pratique les concepts théoriques acquis lors de la formation académique et permet aux entreprises de s'approvisionner d'excellentes ressources qui ont le gout d'apprendre et de contribuer aux succès de l'organisation.

BIBLIOGRAPHIE

1. ADAM: Genomics Formats and Processing Patterns for Cloud Scale Computing. [2013]. En ligne. <https://www2.eecs.berkeley.edu/Pubs/TechRpts/2013/EECS-2013-207.html>. Consulté le 1 novembre 2016.

2. IMPUTE2. [s.d]. En ligne. <https://mathgen.stats.ox.ac.uk/impute/impute_v2.html>. Consulté le 1 novembre 2016.

3. Modernisation d'ADVANCE avec ADAM et Spark. Simon Grondin. [2016]

4. dbSNP: the NCBI database of genetic variation. [2001]. En ligne. <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC29783/>. Consulté le 15 mai 2016.

5. ADVANCE: action in diabetes and vascular disease. [2005]. En ligne. <https://www.ncbi.nlm.nih.gov/pubmed/16075030>. Consulté le 15 mai 2016.

6. CHIAMO. [s.d]. En ligne. <http://innovation.ox.ac.uk/licence-details/chiamo/>. Consulté le 20 novembre 2016.

7. HAPGEN. [2001]. En ligne. <https://mathgen.stats.ox.ac.uk/genetics_software/hapgen/hapgen2.html>. Consulté le 20 novembre 2016.

8. SNPTEST. [s.d.]. En ligne. <https://mathgen.stats.ox.ac.uk/genetics_software/snptest/snptest.html>. Consulté le 20 novembre 2016.

9. GTOOL. [s.d.]. En ligne. <http://www.well.ox.ac.uk/~cfreeman/software/gwas/gtool.html>. Consulté le 20 novembre 2016.

10. GWAS File Formats. [s.d]. En ligne. <http://www.stats.ox.ac.uk/~marchini/software/gwas/file_format.html>. Consulté le 15 mai 2016.

11. Analyse de l'utilisation potentielle de la structure de données de l'Université Berkeley. Liliana ALVARADO MALLMA. [30 novembre 2015].

12. Évaluation et réingénierie du projet GOAT. Victor DUPUY. [15 avril 2016].

13. Oracle VM VirtualBox. [2016]. En ligne. <https://fr.wikipedia.org/wiki/Oracle_VM_VirtualBox>. Consulté le 12 octobre 2015.

14. Oracle VM VirtualBox [s.d]. En ligne. <https://www.virtualbox.org/>. Consulté le 22 mai 2016

15. Linux Ubuntu [s.d]. En ligne. <https://www.ubuntu.com//>. Consulté le 22 mai 2016

16. Linux Ubuntu [2016]. En ligne. <https://fr.wikipedia.org/wiki/Ubuntu>. Consulté le 5 novembre 2016.

17. Python. [s.d]. En ligne. <https://www.python.org/>. Consulté le 15 mai 2016.

18. Python. [2016]. En ligne. <https://fr.wikipedia.org/wiki/Python_(langage)>. Consulté le 7 novembre 2016.

19. PostgreSQL. [s.d]. En ligne. <https://www.postgresql.org/>. Consulté le 15 mai 2016.

20. PostgreSQL. [2016]. En ligne. <https://fr.wikipedia.org/wiki/PostgreSQL>. Consulté le 7 novembre 2016.

21. SBT. [s.d]. En ligne. <http://www.scala-sbt.org/>. Consulté le 15 mai 2016.

22. SBT. [2016]. En ligne. < https://en.wikipedia.org/wiki/SBT_(software) >. Consulté le 7 novembre 2016.

23. Scala. [s.d]. En ligne. <http://www.scala-lang.org/>. Consulté le 23 juillet 2016.

24. Scala. [2016]. En ligne. <https://fr.wikipedia.org/wiki/Scala_(langage) >. Consulté le 7 novembre 2016.

25. Apache Parquet.[s.d]. En ligne. <https://parquet.apache.org/>. Consulté le 23 juillet 2016.

26. Apache Parquet. [2016]. En ligne. <https://en.wikipedia.org/wiki/Apache_Parquet>. Consulté le 5 novembre 2016.

27. Apache Spark. [s.d]. En ligne. <http://spark.apache.org/>. Consulté le 23 juillet 2016.

28. Apache Spark. [2016]. En ligne. <https://fr.wikipedia.org/wiki/Apache_Spark>. Consulté le 5 novembre 2016.

29. Apache Avro.[s.d]. En ligne. <https://avro.apache.org/>. Consulté le 23 juillet 2016.

30. Apache Avro. [2016]. En ligne. <https://en.wikipedia.org/wiki/Apache_Avro>. Consulté le 10 aout 2016.

31. GitHub. [2016]. En ligne. <https://github.com/>. Consulté le 15 mai 2016.

32. GitHub. [2016]. En ligne. <https://fr.wikipedia.org/wiki/GitHub>. Consulté le 7 novembre 2016.

33. Affymetrix. [2016]. En ligne. <http://www.affymetrix.com/catalog/131533/AFFY/Genome-Wide-Human-SNP-Array-6.0#1_1>. Consulté le 7 novembre 2016.

34. Apache Flink. [2016]. En ligne. <https://en.wikipedia.org/wiki/Apache_Flink>. Consulté le 24 novembre 2016.

35. Apache Flink. [2016]. En ligne. <https://flink.apache.org/>. Consulté le 24 novembre 2016.

ANNEXE I

SCRIPT GENTOTSV.PY

```
#!/usr/bin/python
#title           :gentotsv.py
#description     :Script to process impute files .gz and create a csv file
#author          :Diego Alvarez
#date            :2016-06-05
#python_version  :3.5
#==============================================================================
=====

import gzip
import os
import fnmatch
import csv
import sys
import getopt
import time
import linecache
import utils
import config
from multiprocessing import Pool, Process
import multiprocessing
from functools import partial



def script_usage():
  print 'gentotsv.py  -h<help> -t<threads> -s<sourcedir> -
d<destinationdir> -f<samplefile>'
  print '---------'
  print 'If no parameters are passed, default values are taken from
<config.py>'
  print 'Default #threads =  #processor cores'
  print '----------------'

  return


def get_gen_file_columns(p_source_dir,p_source_file):
 with gzip.open(p_source_dir+p_source_file,'rb') as genfile:

   utils.log(logger,"GEN file: "+ p_source_file)

   columns=genfile.readline().split()
   totalcolumns = len(columns)

   utils.log(logger,"Columns in GEN file: "+str(totalcolumns))
```

```
 genfile.close()
 return totalcolumns

def create_sample_file(p_source_dir,p_destination_dir, p_source_file,
p_file_type):

   utils.log(logger,"Begin - create_sample_file -")

   samplecountlines = 0
   source_file = utils.get_file_name(str(p_source_file))

   with open(p_destination_dir+"SAM_"+source_file+p_file_type, 'wb') as
xfile:

    utils.log(logger,"Reading file SAMPLE: " + p_source_file)

    csvwriter = csv.writer(xfile,delimiter='\t',quotechar='"',
quoting=csv.QUOTE_MINIMAL)

    with open(p_source_dir+p_source_file,'rb') as samplefile:

     INDfilelist = []

     for line in samplefile:

          samplecountlines=samplecountlines+1

          if samplecountlines <= 2:

            seq=str(samplecountlines * (-1)).split()
            columns=line.split()
            csvwriter.writerow(seq+columns)

          #Start counting individuals
          if samplecountlines > 2:

              seq=str(samplecountlines-2).split()
              columns=line.split()

              col01= columns[0:2] #to create the file ID
              csvwriter.writerow(seq+columns)

           #Create empty INDIVIDUAL file
              INDfilename = create_individuals_file(p_destination_dir,
seq[0]+"_"+col01[0]+"_"+col01[1], p_file_type)
              #Create list with Individuals file
              INDfilelist.append(INDfilename)


    samplefile.close()
   xfile.close()

   utils.log(logger,"ÉCHANTILLON file lines: "+ str(samplecountlines))
   utils.log(logger,"End - create_sample_file -")
   return INDfilelist
```

```
def create_individuals_sample_files(p_source_dir,p_destination_dir,
p_source_file, p_file_type):
   utils.log(logger,"Begin - create_individuals_sample_files -")

   samplecountlines = 0
   source_file = utils.get_file_name(str(p_source_file))
   INDfilelist = []

   with open(p_source_dir+p_source_file,'rb') as samplefile:
     for line in samplefile:

       samplecountlines = samplecountlines + 1
       columns = line.split()

       if samplecountlines == 1:
         headerline = columns[:]

       elif samplecountlines == 2:

         datatypeline = columns[:]
       else:
         individualline = samplecountlines - 2
         with
open(p_destination_dir+"SAM_"+str(individualline)+"_"+str(columns[0])+"_"+
str(columns[1])+p_file_type, 'wb') as xfile:

                  csvwriter =
csv.writer(xfile,delimiter='\t',quotechar='"', quoting=csv.QUOTE_MINIMAL)

                  for i in range(0, len(columns)):


csvwriter.writerow([headerline[i]]+[datatypeline[i]]+[columns[i]])


                  #Create empty INDIVIDUAL file
                  INDfilename = create_individuals_file(p_destination_dir,
str(individualline)+"_"+columns[0]+"_"+columns[1], p_file_type)
                  #Create list with Individuals file
                  INDfilelist.append(INDfilename)


         xfile.close()
   samplefile.close()

   utils.log(logger,"ÉCHANTILLON file lines: "+ str(samplecountlines))
   utils.log(logger,"End - create_individuals_sample_files -")

   return INDfilelist


def create_snp_file(p_source_dir,p_destination_dir, p_source_file_type,
p_dest_file_type):
```

```
  utils.log(logger,"Begin - Create SNP file -")

  filename = p_destination_dir+"SNP"+p_dest_file_type
  open(filename, 'w').close()

  for file_list in sorted(os.listdir(p_source_dir)):
     if fnmatch.fnmatch(file_list,'*'+p_source_file_type):
       with gzip.open(p_source_dir+file_list,'rb') as genfile:

         sequence=0
         gencountlines=0

         utils.log(logger,"Reading file GEN: " + file_list)

         with open(filename,'ab') as SNPfile:
               csvwriter =
csv.writer(SNPfile,delimiter='\t',quotechar='"',
quoting=csv.QUOTE_MINIMAL)

               #readlines() Loads full .gen file into memory and split in
lines. To many threads
               # or very big files can cause memory overflow.
               #for line in genfile.readlines():
               for line in genfile: #Read file line by line
                   gencountlines=gencountlines+1

                   columns=line.split()
                   col05=columns[0:5]


                   source_file = utils.get_file_name(file_list)

                   sequence=sequence+1
                   seq=str(sequence).split()

                   csvwriter.writerow([source_file]+seq+col05)


         SNPfile.close()
       genfile.close()

  utils.log(logger,"End - Create SNP file -")
  return


def create_individuals_file(p_destination_dir, p_filename, p_file_type):

  filename = p_destination_dir+"IND_"+p_filename+p_file_type

  open(filename, 'w').close()

  return filename
```

```
def
convert_cols_to_lines(p_source_dir,p_source_file,p_destination_dir,p_dest_
file_list, p_individualsposlist, p_gen_column):

           utils.log(logger,"Begin - convert_gen_cols_to_ind_lines - ")
           positionindex = p_individualsposlist.index(p_gen_column)
           regex =
r"^{0}.*{1}$".format(p_destination_dir+"IND_"+str(positionindex+1)+"_",des
tination_file_type)

           p_indfilename = utils.find_file_in_list(p_dest_file_list,regex)
           source_file = utils.get_file_name(str(p_source_file))

           try:

              col = int(p_gen_column)

           except:
              e = sys.exc_info()[0]
              utils.log(logger,e)


           #Open individuals file
           with open(p_indfilename,'a') as indfile:


             utils.log(logger,"Writing IND .tsv file: "+ p_indfilename)

             csvwriter = csv.writer(indfile,delimiter='\t',quotechar='"',
quoting=csv.QUOTE_MINIMAL)
             sequence = 0

             with gzip.open(p_source_dir+p_source_file,'rb') as genfile:
                for line in genfile: #reads line by line .gen file.

                 #readlines() loads full .gen file into memory and split
in lines. To many threads
                 # or very big files can cause memory overflow.
                 #for line in genfile.readlines():

                   sequence=sequence+1

                   seq=str(sequence).split()
                   columns=line.split()


csvwriter.writerow([source_file]+seq+columns[col:col+3])

             indfile.close()

             utils.log(logger,"Lines in source file: "+ str(sequence))

           genfile.close()
           utils.log(logger,"End - convert_gen_cols_to_ind_lines - ")
           return
```

```
def
update_individuals_file(p_source_dir,p_source_file_type,p_destination_dir,
p_dest_file_list):

    utils.log(logger,"Begin - update_individuals_file -")
    for file_list in sorted(os.listdir(p_source_dir)):

     if fnmatch.fnmatch(file_list,'*'+p_source_file_type):

         if  __name__ =='__main__':

                #with gzip.open(p_source_dir+file_list,'rb') as genfile:
                #read only first line
                genfile = gzip.open(p_source_dir+file_list,'rb')
                columns=genfile.readline().split()
                genfile_columns = len(columns)
                genfile.close()

                utils.log(logger, "numthreads: "+str(numthreads))
                pool = Pool(int(numthreads))

                utils.log(logger,"Reading GEN file: "+ file_list)

                index =5
                individualpos = 0
                individualsposlist = []

                #create list with all individuals position
                while(index < genfile_columns):

                          individualsposlist.append(index)
                          index = index + 3

                func =
partial(convert_cols_to_lines,p_source_dir,file_list,p_destination_dir,p_d
est_file_list,individualsposlist)
                pool.map_async(func,individualsposlist).get(9999999)

    utils.log(logger,"End - update_individuals_file -")
    return


#########################################################################
################################
##############################################Main
function###################################################
#########################################################################
################################



try:

     print 'ARGV      :', sys.argv[1:]
```

```
     opts, args = getopt.getopt(sys.argv[1:], 'ht:s:d:f:',
['help=','threads=','sourcedir=','destinationdir=','samplefile='])
     print 'OPTIONS   :', opts

     #Initialization
     help=0

     samplecount=0
     samplecounttotal=0
     gencount=0
     gencounttotal=0
     poscount=0

     #Get défaut values
     source_dir = config.source_dir_oxford
     source_file_type = config.source_file_type_oxford
     destination_dir = config.destination_dir_oxford
     destination_file_type =config.destination_file_type_oxford
     sample_file = config.sample_file_oxford
     sample_file_format = config.sample_file_format_oxford
     numthreads = multiprocessing.cpu_count()

     print "Relative path" + os.path.dirname(os.path.abspath(__file__))


     #Pass the script name to Log
     logger=utils.create_logger("gentotsv")

     start_time = time.time()
     print "Start time: "+time.ctime()
     utils.log(logger, "Start time: "+time.ctime())



     for opt,arg in opts:

           if opt=='-h':
              help = 1
              script_usage()


           elif opt=='-t':

              global numtreads
              numthreads = arg

           elif opt=='-s':

              source_dir = arg

           elif opt=='-d':

              destination_dir = arg

           elif opt=='-f':
```

```
              sample_file = arg

     if help == 0:

          print "Number of threads: "+str(numthreads)
          utils.log(logger, "Number of threads: "+str(numthreads))
          print "Échantillon file format: "+sample_file_format
          utils.log(logger, "Échantillon file format:
"+sample_file_format)
          print "Source directory: "+source_dir
          utils.log(logger, "Source directory: "+source_dir)
          print "Destination directory: "+destination_dir
          utils.log(logger, "Destination directory: "+destination_dir)
          print "Échantillon file name: "+sample_file
          utils.log(logger, "Échantillon file name: "+sample_file)

          if not os.path.exists(source_dir):
             utils.log(logger, "EXCEPTION - Source directory
"+source_dir+" does not exist")
             sys.exit("EXCEPTION - Source directory "+source_dir+" does
not exist")

          #Create destination directory
          try:
             os.makedirs(destination_dir)
          except OSError as err:
             pass


          if os.path.isfile(source_dir+sample_file):

             #----------------------
             #Convert ÉCHANTILLON to TSV
             # 1 file = 1 individual
             #----------------------
             INDfilelist =
create_individuals_sample_files(source_dir,destination_dir,sample_file,des
tination_file_type)

             # 2 threads. Parallel processing.
             if __name__=='__main__':

                 #----------------------
                 #Create SNP file with 1st 5 columns of GEN file
                 #----------------------

#create_snp_file(source_dir,destination_dir,source_file_type,
destination_file_type)
                  p1 = Process(target=create_snp_file,
args=(source_dir,destination_dir,source_file_type, destination_file_type))
                  p1.start()

                 #----------------------
                 #Convert GEN to TSV
```

```
                # 1 file = 1 individual
                #----------------------

#update_individuals_file(source_dir,source_file_type,destination_dir,INDfi
lelist)
                 p2 = Process(target=update_individuals_file,
args=(source_dir,source_file_type,destination_dir,INDfilelist))
                 p2.start()

                 p1.join()
                 p2.join()


          else:
               utils.log(logger," EXCEPTION - Sample File: " + sample_file
+ " does not exists")
               sys.exit("Sample File: " + sample_file + " does not
exists")


     print time.ctime()
     utils.log(logger, "End time: "+time.ctime())

except getopt.GetoptError as err:
  print str(err)
  utils.log(logger,str(err))
  sys.exit(2)
except:
  exc_type, exc_obj, tb = sys.exc_info()
  f = tb.tb_frame
  lineno = tb.tb_lineno
  filename = f.f_code.co_filename
  linecache.checkcache(filename)
  line = linecache.getline(filename, lineno, f.f_globals)
  print 'EXCEPTION IN ({}, LINE {} "{}"): {}'.format(filename, lineno,
line.strip(), exc_obj)
  utils.log(logger,'EXCEPTION IN ({}, LINE {} "{}"): {}'.format(filename,
lineno, line.strip(), exc_obj))
  sys.exit(2)
```

ANNEXE II

SCRIPT EXPORT.PY

```
#!/usr/bin/python
#title           :export.py
#description     :Script generate tsv from Advance, dbSNP and .gen/sample
files
#author          :Simon Grondin - Diego Alvarez
#date            :2016-06-05
#python_version  :3.5
#==============================================================================
=====


import sys
import os
from time import sleep
import psycopg2
import utils
import config
import re



# Constants
# utils.NEWLINE_REPLACEMENT = " " # Can be uncommented and edited
# utils.SUBARRAY_SEPARATOR = ";" # Can be uncommented and edited



if len(sys.argv) != 7:
     print "Passed arguments were: " + str(sys.argv)
     print "Arguments must be: export.py outputDir dbHost dbName dbUser
dbPassword importGenotypeFlag"
     exit(0)


logger=utils.create_logger("export")

output_dir = "./" + sys.argv[1]
print output_dir
output_dir = output_dir if output_dir[-1] == "/" else output_dir + "/"
connection_string = "dbname={1} host={0} user={2}
password={3}".format(*sys.argv[2:])

importGenotypesFlag = None
sourcesDirectory = None


importGenotypesFlag = sys.argv[6]
if importGenotypesFlag == "Y":
```

```
  sourcesDirectory = config.source_converted_oxford
  file_type_oxford = config.destination_file_type_oxford

  if sourcesDirectory[-1]=="/":
     sourcesDirectory = sourcesDirectory
  else:
     sourcesDirectory=sourcesDirectory + "/"

  sourcesFileList = sorted(os.listdir(sourcesDirectory))


print "Connecting to " + connection_string
conn = psycopg2.connect(connection_string)
cursor = conn.cursor()
print "Connected\n"
sleep(2)


#########################
### Individuals_##### ###
#########################
print "Exporting Individuals"

# array_agg + array_to_string = Like SUM(), but for text fields.
Concatenates strings.
cursor.execute("""
      SELECT
            I.id, I.dob, I.sex, I.ethnic_code, I.centre_name,
I.region_name,
            I.country_name, I.comments, COALESCE(B.batches, '')
      FROM "public"."person" I
      LEFT JOIN (
            SELECT B.person_id,
array_to_string(array_agg(B.batch_id::text), '{0}') AS batches
            FROM "public"."batches" B
            GROUP BY B.person_id
            ORDER BY B.person_id ASC
      ) B ON I.id = B.person_id
      ORDER BY I.id ASC
""".format(utils.SUBARRAY_SEPARATOR))
if importGenotypesFlag != "Y":
     print "Writing " + str(cursor.rowcount) + " files\n"


# This section will need to create the headers and the rows dynamically
because
# the number of columns is not known ahead of time! See utils.py for more
info.
# 8 is where the batches subarray is located
subarrayPos = 8
nbBatches = utils.size_of_subarray(cursor, subarrayPos)
batchPlaceholders = utils.make_placeholders(subarrayPos, nbBatches)
headerPlaceholders = utils.make_headers("batch", 0, nbBatches)
f = cursor.fetchone()
current_id = -1
```

```
migratedindividuals=0

utils.log(logger, "Writing <<Individuals>> files")
utils.log(logger, "-----------------------------------")

while f:
     current_id = f[0]

     if importGenotypesFlag == "Y":

                regex =  r"^SAM_[0-
9]+_{0}_.*{1}$".format(current_id,file_type_oxford)
                filename =  utils.find_file_in_list(sourcesFileList,
regex)
                familyID = ''
                fatherID = ''
                motherID = ''

                if filename == None:
                        utils.log(logger, "ID: "+str(current_id) +" -Not
found in SAMPLE file")
                else:

                  os.makedirs(output_dir + str(current_id))

                  with
open("{0}{1}/Individuals_{1}.tsv".format(output_dir, current_id), "w") as
file:

file.write("individualId,familyId,paternalId,maternalId,dateOfBirth,gender
,ethnicCode,centreName,region,country,notes,missingCallFreq,{0}\n".format(
headerPlaceholders))

familyID=utils.get_individual_info(sourcesDirectory+filename, "ID_2")

fatherID=utils.get_individual_info(sourcesDirectory+filename, "father")

motherID=utils.get_individual_info(sourcesDirectory+filename, "mother")

sex=utils.get_individual_info(sourcesDirectory+filename, "sex")

missingCallFreq=utils.get_individual_info(sourcesDirectory+filename,
"missing")
                    migratedindividuals = migratedindividuals + 1

                    #According to Impute2
                    if sex=='1':
                       sex="m"
                    elif sex=='2':
                       sex="f"

                    li = utils.to_prepared_list(f +
(familyID,fatherID,motherID,missingCallFreq))
                    li = utils.break_subarray(li, subarrayPos, nbBatches)
```

```
                    if sex != li[2]:
                         utils.log(logger, "Difference - Sample SEX:
"+sex+ " vs. Advance SEX: "+li[2])



file.write(("""{0},{11},{12},{13},{1},{2},{3},{4},{5},{6},{7},{14},"""+bat
chPlaceholders+"""\n""").format(*li))


     else:
          os.makedirs(output_dir + str(current_id))
          with open("{0}{1}/Individuals_{1}.tsv".format(output_dir,
current_id), "w") as file:

file.write("individualId,familyId,paternalId,maternalId,dateOfBirth,gender
,ethnicCode,centreName,region,country,notes,missingCallFreq,{0}\n".format(
headerPlaceholders))
               migratedindividuals = migratedindividuals + 1
               li = utils.to_prepared_list(f)
               li = utils.break_subarray(li, subarrayPos, nbBatches)


file.write(("""{0},,,,{1},{2},{3},{4},{5},{6},{7},,"""+batchPlaceholders+"
""\n""").format(*li))

     f = cursor.fetchone()

################################
### Individuals_#####_Visits ###
################################
print "Exporting Individuals Visits"
cursor.execute("""
     SELECT V.person_id, V.id, V.visit_date, V.form_id, V.fasting, V.descr
     FROM "public"."visit" AS V
     ORDER BY V.person_id ASC, V.id ASC
""")
if importGenotypesFlag != "Y":
     print "Writing " + str(cursor.rowcount) + " files\n"
f = cursor.fetchone()
current_id = -1

utils.log(logger, "Writing <<Individuals_Visits>> files")
utils.log(logger, "-----------------------------------")

while f:
     if f[0] != current_id: # We only close the file when we switch to a
new Person
          try: # Ugly, but Python doesn't let me create ad hoc mocks like
JS
               file.close()
          except:
               pass

          current_id = f[0]
```

```
          createdfile=0

          if importGenotypesFlag == "Y":

               regex =  r"^SAM_[0-
9]+_{0}_.*{1}$".format(current_id,file_type_oxford)
               filename =  utils.find_file_in_list(sourcesFileList, regex)

               if filename == None:
                    utils.log(logger, "ID: "+str(current_id) +" -Not
found in SAMPLE file")
               else:

                    file =
open("{0}{1}/Individuals_{1}_Visits.tsv".format(output_dir, current_id),
"a")

file.write("visitId,visitDate,studyFormId,fasting,description\n")
                    createdfile=1

          else:
                    file =
open("{0}{1}/Individuals_{1}_Visits.tsv".format(output_dir, current_id),
"a")

file.write("visitId,visitDate,studyFormId,fasting,description\n")
                    createdfile=1

     if createdfile==1:
          li = utils.to_prepared_list(f)
          file.write("""{1},{2},{3},{4},{5}\n""".format(*li))

     f = cursor.fetchone()

####################################
### Individuals_#####_Phenotypes ###
####################################
print "Exporting Individuals Phenotypes"
cursor.execute("""
     SELECT V.person_id, V.id, P.name, M.value, P.um, P.descr,
V.visit_date
     FROM "public"."phenotype" AS P
     LEFT JOIN (
          SELECT * FROM "public"."measure" M
     ) AS M ON P.id = M.phenotype_id
     LEFT JOIN "public"."visit" V ON M.visit_id = V.id
     WHERE V.person_id IS NOT NULL -- There's 1 buggy row, exclude it
     ORDER BY V.person_id ASC
""")
if importGenotypesFlag != "Y":
     print "Writing " + str(cursor.rowcount) + " files\n"

f = cursor.fetchone()
current_id = -1
```

```
utils.log(logger, "Writing <<Individuals_Phenotypes>> files")
utils.log(logger, "---------------------------------------")

while f:
     if f[0] != current_id: # We only close the file when we switch to a
new Person
          try: # Ugly, but Python doesn't let me create ad hoc mocks like
JS
               file.close()
          except:
               pass

          current_id = f[0]
          createdfile=0

          if importGenotypesFlag == "Y":

               regex =  r"^SAM_[0-
9]+_{0}_.*{1}$".format(current_id,file_type_oxford)#check if I I can use
IND file
               filename =  utils.find_file_in_list(sourcesFileList, regex)

               if filename == None:
                              utils.log(logger, "ID: "+str(current_id) +" -
Not found in SAMPLE file")
               else:

                    file =
open("{0}{1}/Individuals_{1}_Phenotypes.tsv".format(output_dir,
current_id), "a")

file.write("visitId,phenotypeType,phenotypeGroup,name,measureDataType,meas
ure,units,description,diagnosisDate\n")
                    createdfile=1
          else:
                    file =
open("{0}{1}/Individuals_{1}_Phenotypes.tsv".format(output_dir,
current_id), "a")

file.write("visitId,phenotypeType,phenotypeGroup,name,measureDataType,meas
ure,units,description,diagnosisDate\n")
                    createdfile=1

     if createdfile==1:
          li = utils.to_prepared_list(f)
          file.write("""{1},,,{2},,{3},{4},{5},{6}\n""".format(*li))

     f = cursor.fetchone()


###################################
### Individuals_#####_Genotypes ###
###################################
print "Exporting Individuals Genotypes"
```

```
mapping = {
     "1": "AA",
     "2": "BB",
     "3": "AB",
     "4": "00",
     "5": "A0", # Doesn't happen in practice
     "6": "B0", # Doesn't happen in practice
     "X": "00",
     "x": "00" # Saves a call to lower()
}

# See Simon_Grondin_PFE.pdf for more information about why there's 2
cursors and how they work together
# It's too much information to write it all here.

cursor1 = conn.cursor()
cursor2 = conn.cursor()
# The first query loads data from the "snp_genotype" table, joined with
the "person" table
print "Executing SQL query 1/2..."
cursor1.execute("""
     SELECT G.person_id, G.genotype_version, P.sample, G.genotype
     FROM "public"."snp_genotype" G
     INNER JOIN "public"."person" P on G.person_id = P.id
     ORDER BY G.person_id ASC, G.genotype_version ASC LIMIT 4
""")

# The second query loads data from the "snp_annotation" table, joined with
the "vcf" table
print "Executing SQL query 2/2..."
cursor2.execute("""
     SELECT
          A.genotype_index, A.annotation_version, A.rs,
          V.chromosome, V.position, (V.position + CHAR_LENGTH(ref)),
          V.ref, V.alt, A.allele_a, A.allele_b
     FROM "public"."snp_annotation" A
     LEFT JOIN "public"."vcf" V ON A.rs = V.rs
     ORDER BY A.annotation_version ASC, A.genotype_index ASC
""")
if importGenotypesFlag != "Y":
     print "Writing " + str(cursor1.rowcount) + " files\n"

f1 = cursor1.fetchone()
f2 = cursor2.fetchone()

utils.log(logger, "Writing <<Individuals_Genotypes>> files")
utils.log(logger, "-----------------------------------")

if importGenotypesFlag != "Y":
     while f1:
          current_id = f1[0]
          current_version = f1[1]
          filename =
"{0}{1}/Individuals_{1}_v{2}_Genotypes.tsv".format(output_dir, current_id,
current_version)
```

```
          print "Writing " + filename
          file = open(filename, "a")

file.write("personId,genotypeVersion,échantillon,letter,rs,chromosome,star
t,end,ref,alt,allele_a,allele_b,genotypeLikelihoods0,genotypeLikelihoods1,
genotypeLikelihoods2\n")

          buffer = []
          while f2 and f2[1] == current_version:
               li1 = utils.to_prepared_list(f1[:3])
               li2 = utils.to_prepared_list(f2[2:])
               letter = mapping[f1[3][f2[0]-1]]

buffer.append("""{0},{1},{2},{3},{4},{5},{6},{7},{8},"{9}",{10},{11},,,\n"
"".format(*(li1+[letter]+li2)))
               f2 = cursor2.fetchone()

          file.write("".join(buffer))
          if f2 is None: # rewind
               cursor2.scroll(0, "absolute")
               f2 = cursor2.fetchone()


          file.close()
          f1 = cursor1.fetchone()


#GEN contains new data, different from Advance, I'm creating a new cursor
with same Individuals as
#indivuals files. It is a complement to the others 2 cursors.-

if importGenotypesFlag == "Y":

     print " "
     print "Import Genotypes? : Y..."

     print "Executing SQL query 3..."
     cursor3 = conn.cursor()
     cursor3.execute("""
               SELECT distinct I.id
               FROM "public"."person" I
               LEFT JOIN (
                    SELECT B.person_id
                    FROM "public"."batches" B
                    GROUP BY B.person_id
                    ORDER BY B.person_id ASC
               ) B ON I.id = B.person_id
               ORDER BY I.id ASC
                         """)

     f3 = cursor3.fetchone()

     regex =  r"^SNP.*{0}$".format(file_type_oxford)
     SNPfilename =  utils.find_file_in_list(sourcesFileList, regex)
```

```
     if SNPfilename == None:
           utils.log(logger, "SNP file not found")
     else:
          while f3:
               current_id = f3[0]
               regex =  r"^IND_[0-
9]+_{0}_.*{1}$".format(current_id,file_type_oxford)

               INDfilename =  utils.find_file_in_list(sourcesFileList,
regex)

               if INDfilename == None:
                  utils.log(logger, "ID: "+str(current_id) +" -Not found
in GEN file")
               else:


                    filename =
"{0}{1}/Individuals_{1}_v{2}_Genotypes.tsv".format(output_dir, current_id,
config.imputed_genotype_version)
                    print "Writing " + filename

                    INDfile = open(filename, "a")


INDfile.write("personId,genotypeVersion,échantillon,letter,rs,chromosome,s
tart,end,ref,alt,allele_a,allele_b,genotypeLikelihoods0,genotypeLikelihood
s1,genotypeLikelihoods2\n")

                    with open(sourcesDirectory+SNPfilename,'rb') as
SFile,open(sourcesDirectory+INDfilename,'rb') as IFile:
                         for x, y in zip(SFile,IFile):

                              SNPcolumns=x.split()
                              INDcolumns=y.split()

                              li =
utils.to_prepared_list((str(current_id),config.imputed_genotype_version,SN
Pcolumns[2],SNPcolumns[3],SNPcolumns[4],str(int(SNPcolumns[4])+len(SNPcolu
mns[5])),SNPcolumns[5],SNPcolumns[6],INDcolumns[2],INDcolumns[3],INDcolumn
s[4]))

INDfile.write("""{0},{1},,,{3},{2},{4},{5},{6},{7},{6},{7},{8},{9},{10}\n"
"".format(*li))

                    INDfile.close()
               f3 = cursor3.fetchone()


###############
### Batches ###
###############
print "Exporting Batches"
utils.log(logger, "Writing <<Batches>> file")
utils.log(logger, "-----------------------")
```

```
with open(output_dir + "Batches.tsv", "w") as file:
     file.write("batchId,batchDate,batchType,description,studyName\n")

     # First do the batch table
     cursor.execute("""SELECT B.id, B.batch_date, B.description FROM
"public"."batch" AS B""")
     f = cursor.fetchone()
     while f:
          li = utils.to_prepared_list(f)
          file.write("""{0},{1},ADVANCE-ON,{2},Genotyping
Batch\n""".format(*li))
          f = cursor.fetchone()

     # Then do the échantillons table
     cursor.execute("""SELECT B.id, B.sample_date, B.description FROM
"public"."échantillons" AS B""")
     f = cursor.fetchone()
     while f:
          li = utils.to_prepared_list(f)
          file.write("""S{0},{1},ADVANCE-ON,{2},Enrollment
Batch\n""".format(*li)) # Note the "S" before the Sample ID
          f = cursor.fetchone()
print "------------------------------"
print 'Migrated Individuals: '+str(migratedindividuals)
utils.log(logger, 'Migrated Individuals: '+str(migratedindividuals))
print "Done"
```

ANNEXE III

SCRIPT DIRECTORYHANDLER.SCALA

```
import com.opencsv._
import java.io._
import scala.util.Try
import scala.collection.JavaConversions._
import org.bdgenomics.formats.avro._
import org.slf4j.LoggerFactory

class DirectoryHandler(prefix: String, genotypeVersion: String,
assemblyName: String,
  batchData: collection.mutable.HashMap[String,Batch], dateFormat:
java.text.SimpleDateFormat) {

  private val logger = LoggerFactory.getLogger(this.getClass)
  // Number of columns in the Individual.csv file. Everything past that
number is a batch ID
  val NB_INDIVIDUAL_COLUMNS = 11
  // Extract the type of file
  val filePattern = """^Individuals_([0-9]+)(_(.*))?\.tsv$""".r
  // Extract the Genotype version number
  //val genotypePattern = """^Individuals_([0-9]+)_(v[0-
9]+)_Genotypes\.tsv$""".r
  val genotypePattern = """^Individuals_([0-9]+)_(v[0-
9]+i*)_Genotypes\.tsv$""".r

  // Run a function without having to care if it succeeds or fails
  def ignore(str:String, possibleFailure: () => Unit) =
    try {
      possibleFailure()
    } catch {
      case e : Throwable =>
        //logger.warn(str+" - "+e.getMessage) commented out because each
warning is an object. No enough memory to support them all
      None
    }

  // This is the main function in this file
  def processIndividual(id: String) : Option[Individual] = {
    // List of files in the sub-folder
    val ls = new java.io.File(prefix + id).listFiles.map(file =>
file.getName).sorted
    // Structure to hold incomplete data
    val record = IndividualRecord()

    ls.foreach({ fileName =>
      filePattern.findAllIn(fileName).matchData foreach { m =>

        // Needs an interator or it'll run out of RAM
```

```
        val reader = new CSVIterator(new CSVReader(new FileReader(prefix +
id + "/" + fileName)))
        reader.next() // Skip headers

        m.group(3) match {
          // The current file is for the Visits
          case ("Visits") =>
            reader.foreach { row =>
              val visit = Visit.newBuilder()
              visit.setVisitId(row(0))
              ignore(id+"-"+"setVisitDateEpoch",() =>
visit.setVisitDateEpoch(dateFormat.parse(row(1)).getTime))
              visit.setStudyId(row(2))

visit.setIsFasting(Try(java.lang.Boolean.valueOf(row(3))).getOrElse(false)
)
              visit.setDescription(row(4))
              record.visits = visit :: record.visits // Each row in the
csv adds a visit to the list
            }
          // The current file is for the Phenotypes
          case ("Phenotypes") =>
            reader.foreach { row =>
              val phenotype = Phenotype.newBuilder()
              phenotype.setPhenotypeType(row(1))
              phenotype.setPhenotypeGroup(row(2))
              phenotype.setName(row(3))

              //begin DMA
              //ignore(() =>
phenotype.setMeasureDataType(MeasureDataType.valueOf(row(4))))
            if(row(4) == "") None//
phenotype.setMeasureDataType(MeasureDataType.valueOf(Unknown))
            else ignore(id+"-"+"setMeasureDataType",() =>
phenotype.setMeasureDataType(MeasureDataType.valueOf(row(4))))
            //end DMA

              phenotype.setMeasure(row(5))
              phenotype.setMeasureUnits(row(6))
              phenotype.setDescription(row(7))
            //begin DMA
            if(row(8) =="") None
              else ignore(id+"-"+"setDiagnosisDateEpoch",() =>
phenotype.setDiagnosisDateEpoch(dateFormat.parse(row(8)).getTime))
             //end DMA
              record.phenotypes = phenotype :: record.phenotypes // Each
row in the csv adds a phenotype to the list
            }
          // The current file is for the Individual
          case (null) =>
            reader.foreach { row =>

              val gender = if (Character.toLowerCase(row(5).charAt(0)) ==
'm') Gender.Male
```

```
                else if (Character.toLowerCase(row(5).charAt(0)) == 'f')
Gender.Female
                else Gender.Unknown
              val individual = Individual.newBuilder()
              individual.setGender(gender)
              individual.setIndividualId(row(0))
              individual.setFamilyId(row(1))
              individual.setFatherId(row(2))
              individual.setMotherId(row(3))
              ignore(id+"-"+"setBirthDateEpoch",() =>
individual.setBirthDateEpoch(dateFormat.parse(row(4)).getTime))
              individual.setEthnicCode(row(6))
              individual.setCentreName(row(7))
              individual.setRegion(row(8))
              individual.setCountry(row(9))
              individual.setNotes(row(10))

              //dma
              if(row(11) == "") None
            else ignore(id+"-"+"setMissingCallFreq",()
=>individual.setMissingCallFreq(java.lang.Float.parseFloat(row(11))))
              //dma

              // Ignore the first NB_INDIVIDUAL_COLUMNS
              // Map the remaining columns (batch IDs) with the HashMap of
batch data by ID
              // Flatten: List[Option[T]] -> List[T]
              // Flatten handles both when fields are null and when the
batch ID can't be mapped
              val listBatches =
row.toList.drop(NB_INDIVIDUAL_COLUMNS).map({batchId =>
                if (batchId == "") None else batchData.get(batchId)
              }).flatten
              individual.setBatches(listBatches)
              record.individual = Some(individual)
            }
          // The current file is for the Genotypes
          case (_) =>
            // Extract the genotype version number
            genotypePattern.findAllIn(fileName).matchData foreach { n =>
              // We only want to continue when it's the right version
              if (n.group(2) == genotypeVersion) reader.foreach { row =>
                val contig = Contig.newBuilder()
                contig.setContigName(row(5)) // chromosome
                contig.setAssembly(assemblyName)
                contig.setSpecies("human")

                // DBSNP data
                val variant = Variant.newBuilder()
                variant.setContig(contig.build())
                ignore(id+"-"+"setStart",() =>
variant.setStart(java.lang.Long.parseLong(row(6))))
                ignore(id+"-"+"setEnd",() =>
variant.setEnd(java.lang.Long.parseLong(row(7))))
                variant.setReferenceAllele(row(8))
```

```
                variant.setAlternateAllele(row(9))

                val alleles: List[GenotypeAllele] = List() // List of
GenotypeAllele Ref, Alt, OtherAlt, NoCall

                val genotype = Genotype.newBuilder()

                genotype.setVariant(variant.build())
                // genotype.setSampleId(row(2)) // NULL for now. Leaving
this commented, as documentation
                genotype.setAlleles(alleles)


            //begin dma
                //val likelihoods: List[java.lang.Float] =
List(row(12).toFloat,row(13).toFloat,row(14).toFloat) // List of
Likelihoods
            if (row(12) == "" && row(13) == "" && row(14) == "") None
            else {
                  try {
                        val likelihoods: List[java.lang.Float] =
List(row(12).toFloat,row(13).toFloat,row(14).toFloat) // List of
Likelihoods
                        genotype.setGenotypeLikelihoods(likelihoods)
                      } catch {
                        case e : Throwable =>
                        logger.warn(row(5) + " - " +e.getMessage)
                      }
            }

                genotype.setVariantIdentifier(row(4))

                //end dma

                record.genotypes = genotype :: record.genotypes // Each
row in the csv adds a genotype to the list
              }
            }
        }
      }
    })

    // Only generate an ADAM object when all the data is there
    if (record.isReady) {
      Some(record.toWritable)
    } else {
      None
    }

  }
}
```

ANNEXE IV

CARTOGRAFIE ADAM

Cartographie des données (data mapping) entre les fichiers « .gen » et « .sample » et le format ADAM.

ADAM		.gen/.sample	Commentaires
individualID	String	.sample.ID_1	
familyID	String	.sample.ID_2	
fatherID	String	.sample.father	
motherID	String	.sample.mother	
missingCallFreq	Float	.sample.missing	
visit[0].phenotypes[0].name	String	.sample. "header column"	Male, Female, Unknown
visit[0].phenotypes[0].phenotypeType	String	Constant	B => Boolean D => Decimal C => Integer P => Integer
visit[0].phenotypes[0].measure	String	.sample. (columns 4-N)	
visit[0].phenotypes[0].measureDataType	ENUM	Constant	B => Boolean D => Decimal C => Integer P => Integer
variantIdentifier	String	.gen.RS_ID	
variant.contig.contigName	String	.gen.snpID	snpID ou chromosome#
variant.start	Long	.gen.base-pair position	
variant.end	Long	variant.start + LEN(refAllele)	
allelesDetails[0]	string	.gen.Allele A	
allelesDetails[1]	string	.gen .Allele B	
genotypeLikelihoods[0]	float	.gen.probabilityHomozygoteA1	%
genotypeLikelihoods[1]	float	.gen.probabilityHeterozygote2	%
genotypeLikelihoods[2]	float	.gen.probabilityHomozygoteA3	%

ANNEXE V

SCHEMA PARQUET

```
|-- genotypes: array (nullable = false)
 |    |-- element: struct (containsNull = false)
 |    |    |-- variantIdentifier: string (nullable = true)
 |    |    |-- variant: struct (nullable = true)
 |    |    |    |-- variantErrorProbability: integer (nullable = true)
 |    |    |    |-- contig: struct (nullable = true)
 |    |    |    |    |-- contigName: string (nullable = true)
 |    |    |    |    |-- contigLength: long (nullable = true)
 |    |    |    |    |-- contigMD5: string (nullable = true)
 |    |    |    |    |-- referenceURL: string (nullable = true)
 |    |    |    |    |-- assembly: string (nullable = true)
 |    |    |    |    |-- species: string (nullable = true)
 |    |    |    |    |-- referenceIndex: integer (nullable = true)
 |    |    |    |-- start: long (nullable = true)
 |    |    |    |-- end: long (nullable = true)
 |    |    |    |-- referenceAllele: string (nullable = true)
 |    |    |    |-- alternateAllele: string (nullable = true)
 |    |    |    |-- svAllele: struct (nullable = true)
 |    |    |    |    |-- type: string (nullable = true)
 |    |    |    |    |-- assembly: string (nullable = true)
 |    |    |    |    |-- precise: boolean (nullable = true)
 |    |    |    |    |-- startWindow: integer (nullable = true)
 |    |    |    |    |-- endWindow: integer (nullable = true)
 |    |    |    |-- isSomatic: boolean (nullable = true)
 |    |    |-- variantCallingAnnotations: struct (nullable = true)
 |    |    |    |-- variantIsPassing: boolean (nullable = true)
 |    |    |    |-- variantFilters: array (nullable = false)
 |    |    |    |    |-- element: string (containsNull = false)
 |    |    |    |-- downsampled: boolean (nullable = true)
 |    |    |    |-- baseQRankSum: float (nullable = true)
 |    |    |    |-- fisherStrandBiasPValue: float (nullable = true)
 |    |    |    |-- rmsMapQ: float (nullable = true)
 |    |    |    |-- mapq0Reads: integer (nullable = true)
 |    |    |    |-- mqRankSum: float (nullable = true)
 |    |    |    |-- readPositionRankSum: float (nullable = true)
 |    |    |    |-- genotypePriors: array (nullable = false)
 |    |    |    |    |-- element: float (containsNull = false)
 |    |    |    |-- genotypePosteriors: array (nullable = false)
 |    |    |    |    |-- element: float (containsNull = false)
```

```
|    |    |    |-- vqslod: float (nullable = true)
|    |    |    |-- culprit: string (nullable = true)
|    |    |    |-- attributes: map (nullable = false)
|    |    |    |    |-- key: string
|    |    |    |    |-- value: string (valueContainsNull = false)
|    |    |-- sampleId: string (nullable = true)
|    |    |-- sampleDescription: string (nullable = true)
|    |    |-- processingDescription: string (nullable = true)
|    |    |-- alleles: array (nullable = false)
|    |    |    |-- element: string (containsNull = false)
|    |    |-- expectedAlleleDosage: float (nullable = true)
|    |    |-- referenceReadDepth: integer (nullable = true)
|    |    |-- alternateReadDepth: integer (nullable = true)
|    |    |-- readDepth: integer (nullable = true)
|    |    |-- minReadDepth: integer (nullable = true)
|    |    |-- genotypeQuality: integer (nullable = true)
|    |    |-- genotypeLikelihoods: array (nullable = false)
|    |    |    |-- element: float (containsNull = false)
|    |    |-- nonReferenceLikelihoods: array (nullable = false)
|    |    |    |-- element: float (containsNull = false)
|    |    |-- strandBiasComponents: array (nullable = false)
|    |    |    |-- element: integer (containsNull = false)
|    |    |-- splitFromMultiAllelic: boolean (nullable = true)
|    |    |-- isPhased: boolean (nullable = true)
|    |    |-- phaseSetId: integer (nullable = true)
|    |    |-- phaseQuality: integer (nullable = true)
|-- gender: string (nullable = false)
|-- individualId: string (nullable = true)
|-- familyId: string (nullable = true)
|-- fatherId: string (nullable = true)
|-- motherId: string (nullable = true)
|-- birthDateEpoch: long (nullable = true)
|-- ethnicCode: string (nullable = true)
|-- centreName: string (nullable = true)
|-- region: string (nullable = true)
|-- country: string (nullable = true)
|-- notes: string (nullable = true)
|-- missingCallFreq: float (nullable = true)
|-- visits: array (nullable = false)
|    |-- element: struct (containsNull = false)
|    |    |-- visitId: string (nullable = true)
|    |    |-- visitDateEpoch: long (nullable = true)
|    |    |-- studyId: string (nullable = true)
|    |    |-- isFasting: boolean (nullable = true)
|    |    |-- description: string (nullable = true)
```

```
|    |    |-- phenotypes: array (nullable = false)
|    |    |    |-- element: struct (containsNull = false)
|    |    |    |    |-- name: string (nullable = true)
|    |    |    |    |-- phenotypeGroup: string (nullable = true)
|    |    |    |    |-- phenotypeType: string (nullable = true)
|    |    |    |    |-- measure: string (nullable = true)
|    |    |    |    |-- measureDataType: string (nullable = false)
|    |    |    |    |-- measureUnits: string (nullable = true)
|    |    |    |    |-- description: string (nullable = true)
|    |    |    |    |-- diagnosisDateEpoch: long (nullable = true)
|    |-- batches: array (nullable = false)
|    |    |-- element: struct (containsNull = false)
|    |    |    |-- batchDateEpoch: long (nullable = true)
|    |    |    |-- batchType: string (nullable = true)
|    |    |    |-- description: string (nullable = true)
```

ANNEXE VI

LISTE DE PREREQUIS

Ceci représente les étapes que j'ai dû effectuer pour reproduire l'adaptation d'ADVANCE et dbSNP en format ADAM.

1. Installer *pip*, gestionnaire de paquet utilisé pour installer et gérer des paquets écrits en Python.
    - sudo apt-get install python-pip

2. Installer *PostgreSQL*.
    - sudo apt-get install postgresql

3. Installer *python-psycopg2*, adaptateur PostgreSQL pour Python
    - sudo apt-get install python-psycopg2

4. Créer l'usager *prognomix*
    - sudo -su postgres
    - psql -c "CREATE USER prognomix WITH PASSWORD 'xxxxyyyzzz';"

5. Créer la base de données *advancedb*
    - sudo -su postgres
    - psql -c "CREATE DATABASE advancedb OWNER prognomix;"
    - psql advancedb < Advance_prognomix_pgsql_dump_backup.sql

6. Soit télécharger la BD dbSNP human (Build 142) du NCBI et rouler les scripts "prepareVCF.py" et "loadvcf.py",
    - http://www.ncbi.nlm.nih.gov/news/10-17-2014-dbsnp-human-build-142/

    ou se connecter à PostgreSQL et créer la table « vcf » sans données :

    - sudo -u postgres psql advancedb

- psql -c "CREATE TABLE public.vcf
  (
  id integer NOT NULL,
  chromosome text NOT NULL,
  "position" integer NOT NULL,
  rs character varying(32) NOT NULL,
  ref text NOT NULL,
  alt text NOT NULL,
  quality text,
  filter text,
  info text,
  CONSTRAINT vcf_pk PRIMARY KEY (id)
  );"

- GRANT ALL PRIVILEGES ON TABLE vcf TO public;
- ALTER TABLE public.vcf OWNER TO *prognomix*;

Note: les scripts prepareVCF.py et loadVCF.py vont chercher les fichiers source dans le disque courant où se trouvent les scripts (*relative path*). Soit placer les sources dans le même disque ou modifier les scripts.

7. Installer *Java*
   - sudo apt-get install oracle-java8-installer
8. Installer *Scala* et *sbt*

   - sudo apt-get remove scala-library scala
   - sudo wget www.scala-lang.org/files/archive/scala-2.11.8.deb
   - sudo apt-get -f install
   - sudo dpkg -i scala-2.11.8.deb

- sudo apt-get update
- sudo apt-get install scala
- sudo wget https://bintray.com/artifact/download/sbt/debian/sbt-0.13.6.deb
- sudo dpkg -i sbt-0.13.6.deb
- sudo apt-get update
- sudo apt-get install sbt

9. Vérifier si *Java* et *Scala* se sont bien installées

    - Java –version
    - Scala –version

10. Installer git.
    - sudo apt-get install git

11. Installer Spark
    - wget http://d3kbcqa49mib13.cloudfront.net/spark-1.6.2.tgz

12. Décompresser le fichier:
    - tar xvf spark-1.6.2.tgz

13. Copier les sources dans un autre répertoire.
    - Ex. cp -r spark-1.6.2 /usr/bin

14. Modifier le PATH des sources Spark : ajouter dans fichier de config /home/. bashrc les lignes suivantes

    - export PATH=$PATH:/usr/bin/spark-1.6.2/bin (le path ou vous avez copié les sources Spark, étape 11).

- export SPARK_HOME=/usr/bin/spark-1.6.2 (le path ou vous avez copié les sources Spark, étape 11).

15. Exécuter le fichier de config
    - source ~/.bashrc

16. Lancer l'installation de Spark :
    - cd spark-1.6.2
    - ./sbt/sbt assembly

17. Tester l'installation de Spark : lancer la commande suivante.
    - /bin/spark-shell (ou aller dans /usr/bin/spark-1.6.2/bin)

18. Tester Spark UI
    - http://10.0.2.15:4040/jobs

Pour convertir le schéma Avro en Java classes

19. Telecharcher la version 1.7.7 d'Avro, seulement les fichiers *avro-1.7.7.jar* et *avro-tools-1.7.7.jar*
    - http://apache.claz.org/avro/avro-1.7.7/java/

20. Cloner avec GitHub le depot *bdg-formats*, la branche *crchum*
    - https://github.com/bigdatagenomics/bdg-formats/ (crchum fork)

21. Ajouter les nouveaux champs souhaités dans le fichier:
    - */bdg-formats/src/main/resources/avro/bdg.avdl*

22. Convertir le fichier *advl* en *avpr*. Exécuter la commande suivante:
    - java -jar avro-tools-1.7.7.jar idl bdg.avdl bdg.avpr

23. Generer les classes Java:
    - java -jar avro-tools-1.7.7.jar compile protocol bdg.avpr .

24. Vérifier si les classes ont été créés dans:

- /org/bdgenomics/formats/avro/

ANNEXE VII

COMMANDES POUR LANCER LES SCRIPTS

prepareVCF.py:

- python prepareVCF.py /repertoire_des_sources/All_20160527.vcf

loadVCF.py:

- python loadVCF.py All_20160527.vcf.csv 127.0.0.1 advancedb postgres *mdp_de_postgres*

gentotsv.py:

- python gentotsv.py (prend les valeurs par défaut du script config.py)
- gentotsv.py -h[help] -t[threads] -s[sourcedir] -d[destinationdir] -f[samplefile] -> (en passant des parametres)

export.py:

- python export.py {destination_des_tsv} {ip_postgresql_server} {schema_postgresql} {owner_schema_postresql} {password_owner_schema}{import_genotypes_flag}
- Ex. python export.py "/Test/output" 127.0.0.1 advancedb prognomix *mdp_de_prognomix* Y

Sbt:

- Lancer SBT : sbt -mem 4096
- Compiler projet (dans SBT) : compile
- Faire le menage du repertoire *target* directory (dans SBT) :clean
- Executer le projet (dans SBT) : run {genotype_version} {dbSNP_version} {nbThreads}. Ex. run v2i NCI142 2

Spark :

- Demarrer Spark: spark-shell --verbose --driver-memory 6g