ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

RAPPORT DE PROJET DE 15 CREDITS PRÉSENTÉ À
L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

COMME EXIGENCE PARTIELLE
À L'OBTENTION DU DIPLÔME D'ÉTUDES SUPÉRIEURES SPÉCIALISÉES
EN TECHNOLOGIES DE L'INFORMATION

PAR
Ivan KIZEMA

APPLICATION DES TECHNOLOGIES BIG DATA DANS LE DOMAINE D'ANALYSE GÉNÉTIQUE

MONTRÉAL, LE 30 NOVEMBRE 2015

PRÉSENTATION DU JURY

CE RAPPORT DE PROJET A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE :

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

Professeur Alain Abran, président du jury
Département de génie logiciel et TI à l'École de technologie supérieure

REMERCIEMENTS

Avant tout, je souhaite remercier mon directeur du projet, Prof. Alain April. Je le remercie de m'avoir donné l'opportunité d'enrichir mes connaissances dans les domaines des technologies Big Data et d'analyses génétiques.

Je souhaite également remercier M. David Lauzon, étudiant en Ph.D. et chargé de cours à ÉTS, de m'avoir intégré et guidé dans le projet.

Leurs encadrements et conseils constructifs ont permis une montée rapide en compétences, essentielle dans la réalisation du projet.

De même, je souhaite remercier l'ensemble des équipes des bio-informaticiens de CRCHUM impliqués sur le projet.

APPLICATION DES TECHNOLOGIES BIG DATA DANS LE DOMAINE D'ANALYSE GÉNÉTIQUE

Ivan KIZEMA

RÉSUMÉ

Le domaine de la génétique se développe rapidement et les chercheurs doivent faire face aux quantités des données de plus en plus importantes. Les outils actuellement utilisés montrent des signes de faiblesse face à cette quantité des données et une solution doit être trouvée.

Le projet a pour l'objectif d'accompagner les bio-informaticiens dans cette démarche de traitement des données à grande échelle à l'aide des technologies Big Data. En partenariat avec une équipe des chercheurs de CRCHUM, le projet développe un scénario de cas d'application permettant de couvrir le domaine de génotypage.

Le projet présente des enjeux multiples dont notamment deux enjeux essentiels. Le premier enjeu est la recherche de normalisation et de modélisation des données et des processus pour le domaine de génotypage. Le deuxième enjeu est l'optimisation de la rapidité des calculs sur les données en masse.

La preuve de concept adam-ibs, développée dans le cadre du projet, permet de valider la faisabilité de la modélisation des données couvrant les besoins spécifiques du domaine de génotypage. De même, adam-ibs présente la méthodologie permettant d'adapter les modèles et les processus aux nouvelles implémentations dans le contexte Big Data.

APPLICATION OF BIG DATA TECHNOLOGIES IN THE FIELD OF GENETIC ANALYSIS

Ivan KIZEMA

ABSTRACT

The field of genetics is developing rapidly and researchers have to face the fast growing amount of data. Currently used tools show signs of weakness against the amount of data and a solution must be found.

The project's goal is to guide bioinformaticians in large-scale data processing approach using Big Data technologies. In partnership with a team of researchers of the CHUM Research Centre, the project develops an application case scenario to cover genotyping field.

In addition to optimisation of calculation speed on mass data, the project covers the issue concerning the lack of standardisation in the field of genotyping. The developed proof of concept adam-ibs introduces data and processes normalization methodologies to cover specific genotyping need in Big Data context.

TABLE DES MATIÈRES

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

ADN - Acide désoxyribonucléique

BAM - Binary Alignment Map (Version binaire du format SAM)

DSL - Domain Specific Language

ETL - Extract Transform Load

IaaS - Infrastructure as a Service

IBD - Identical By State

IBS - Identical By Descent

MDS - MultiDimensional Scaling

OLAP - OnLine Analytical Processing

PaaS - Platform as a Service

SaaS - Software as a Service

SAM - Sequence Alignment Map

VCF - Variant Call Format

WGAS - Whole Genome Association Studies

WGS - Whole Genome Shotgun

XML - Extensible Markup Language

INTRODUCTION

Le domaine de la génétique se développe continuellement et la quantité de données n'arrête pas de croître. Les volumes de données rendent leur traitement difficile par les méthodes classiques. La tendance à l'accroissement des quantités de données n'est pas prête à être renversée et une solution doit être trouvée.

L'objectif du projet est d'accompagner les bio-informaticiens et les chercheurs en génétique dans cette phase de croissance des données. Pour ce faire, le projet tentera de leur fournir des outils de traitement performants basés sur les technologies Big Data. Afin d'y parvenir, les enjeux et les difficultés du projet sont multiples. Il s'agit de comprendre le contexte et les besoins en génotypage afin de proposer une solution complète et durable.

Actuellement, le domaine de génotypage est couvert par une multitude de petites équipes. De ce fait, chaque équipe utilise ses méthodes de travail, sa modélisation des données et ses résultats. Ainsi, l'aboutissement à une solution durable sous-entend la nécessité d'aboutir à une solution universelle et standardisée. L'un des prérequis est donc de proposer un modèle de données qui couvre l'ensemble des besoins en analyse génétique : un modèle qui serait une base commune la plus complète et qui permettrait l'évolution vers les nouvelles fonctionnalités tout en gardant la compatibilité avec les versions antérieures.

Le présent document est divisé en trois parties distinctes. La première partie est une revue de littérature qui présentera les initiatives existantes en traitement des données en masse ainsi que la logique utilisée en traitement des données génétiques. La seconde partie se penchera plus en détail sur l'outil cible d'analyse génétique utilisant les technologies Big Data. Dans un premier temps, cette partie présentera la vision macroscopique de la plateforme. Puis, elle présentera le détail de migration et d'implémentation de la logique de traitement génétique inspirée de l'outil Plink largement utilisé aujourd'hui dans le domaine. Enfin, la troisième partie présentera les résultats obtenus et le potentiel d'évolution du projet.

CHAPITRE 1

LA REVUE BIBLIOGRAPHIQUE

1.1 Introduction

Le projet vise à apporter des solutions de traitement de données à grande échelle dans le domaine de l'analyse génétique et plus spécifiquement le génotypage. Afin de couvrir le sujet dans sa globalité, ce premier chapitre est divisé en deux parties. La première partie présentera la synthèse de l'état de l'art de ce qui se fait dans la matière de traitement des données à grande échelle dans le domaine des analyses génétiques. La deuxième partie s'intéressera plus spécifiquement aux différents aspects d'analyses génétiques et notamment le domaine de génotypage.

1.2 Le traitement des données génétiques à grande échelle

De même que dans les autres industries, le secteur de la génomique s'intéresse de plus en plus à l'infonuagique (c.-à-d. de l'anglais « cloud computing ») comme le constate Vivien Marx [1]. Parmi les nombreux avantages que pourrait apporter l'infonuagique publique ou privée, nous pouvons notamment parler de la centralisation et de l'optimisation de l'utilisation des ressources TI. De même, ces derniers favorisent la mise en place des outils de traitement de données plus complexes, tels que des systèmes repartis permettant le traitement des données en masse ou communément appelé Big Data.

Dans l'article « Big data, Hadoop and cloud computing in genomics » [2], les auteurs soutiennent l'idée que les plateformes infonuagiques favorisent la mise en place des solutions Big Data. En effet, au-delà des avantages immédiats apportés par la centralisation et l'optimisation des ressources informatiques, les solutions Big Data sont souvent accessibles et proposées dans les offres PaaS (Platform as a Service) telles qu'Amazon EC2 et Windows Azure.

Également, plusieurs initiatives d'offres SaaS ont récemment vu le jour dans le domaine de la génomique. Le Cloud BioLinux [3], créé par le J. Craig Venter Institute, est une offre SaaS pour la génomique hébergée sur Amazon EC2. Ce service implémente une centaine d'outils d'analyse génétique et dispose d'une interface graphique disponible sur le Web pour les utilisateurs. L'offre SaaS Atlas2 Cloud [4] propose, elle aussi, un cadriciel qui permet l'analyse génomique basée sur une étude de cas. Les auteurs soulignent que l'infrastructure infonuagique optimisée et l'interface graphique Web favorisent et facilitent l'utilisation du service. Cependant, afin d'être facilement réutilisés dans d'autres projets, ces logiciels, les modèles sous-jacents et les processus d'utilisation doivent être standardisés et généralisés.

On voit également apparaître les nouveaux projets tels que Galaxy [5]. Galaxy est un outil, disponible en logiciel libre, fournissant une plateforme Web pour la recherche biomédicale. Cette plateforme pouvant facilement être déployée, dans un environnement de nuage privé ou public, peut aussi être disponible localement. Les travaux d'amélioration de cette solution, présentés dans « Cloud-based bioinformatics workflow platform for large-scale next-generation sequencing analyses » [6] s'appuie sur le flux de travail proposé initialement par Galaxy et l'améliore par la proposition de fonctionnalités enrichies. Notamment, cette publication intègre : les outils de gestion de grandes quantités de données (c.-à-d. le Big Data), les systèmes de déploiement automatiques sur les infrastructures infonuagiques avec l'auto dimensionnement, ainsi qu'une bibliothèque d'outils spécifiques aux différents domaines de la génomique. Par ailleurs, on constate que le caractère libre de ces solutions permet d'engendrer ce qui est appelé de l'innovation ouverte et rapide. L'approche de collaboration du logiciel libre bouleverse le modèle économique des firmes privées en permettant à des équipes virtuelles et distribuées, sans nécessiter de grand capital, de faire des avancées rapides et offrir des logiciels complexes et utiles à l'industrie.

Le projet Globus Genomics [7] de l'Université de Chicago propose aussi une solution SaaS pour l'analyse complète de séquençage du génome. Ce service ne permet pas encore de déploiement et de personnalisation automatique de l'espace de travail pour les projets. Toutefois, il propose une démarche d'accompagnement. L'article « A case study for cloud

based high throughput analysis of NGS data using the globus genomics system » [8] présente un cas d'utilisation de la plateforme dans le cadre d'une analyse de séquençage du génome humain. Notamment, ce logiciel permet de traiter les formats des fichiers standardisés du domaine de séquençage (c.-à-d. SAM, BAM et VCF), de définir un flux de travail personnalisé au projet et d'effectuer les traitements associés à chaque étape du séquençage.

En 2014, l'Université de Californie Berkeley lance un projet de logiciel libre intitulé ADAM [9]. Ce projet vise à améliorer et à normaliser les modèles des données en génomique. Il vise aussi à proposer des patrons de traitement de grandes quantités de données sur les systèmes infonuagiques distribués Big Data. Ce cadriciel de traitement moderne utilise la toute nouvelle technologie Spark [10]. Spark permet d'effectuer l'analyse des données à grande échelle d'une manière distribuée et interactive qui peut être implémentée par-dessus de la technologie bien connue de Hadoop et de HDFS [11]. Grâce à ADAM, la persistance des données s'effectue avec les technologies de sérialisation Apache Avro [12] et celles de stockage en colonne immuable d'Apache nommée Parquet [13]. ADAM se concentre notamment sur le séquençage génétique et propose des fonctionnalités et des traitements adaptés et très performants. Notamment, ADAM propose un modèle de données normalisé pour répondre aux besoins de séquençage en intégrant/remplaçant tous les besoins des formats standardisés SAM et VCF.

1.3 Les domaines de l'analyse génétique

Le domaine des analyses génétiques est très vaste. On peut notamment distinguer deux domaines plus spécifiques et populaires de séquençage et de génotypage.

1.3.1 Le séquençage génétique

Le domaine de séquençage, comme son nom l'indique, se concentre sur la détermination de la séquence d'ADN. Le domaine vise à déterminer la séquence des gènes, des chromosomes et voir du génome complet des êtres vivants. La technique couramment utilisée pour séquencer un génome est le « Whole Genome Shotgun (WGS) » [14]. Cette méthode consiste

à casser l'ADN en segments, déterminer la séquence de chaque segment et reconstituer la séquence globale en recomposant les séquences des différents segments. Elle requiert beaucoup de puissance de calcul afin de reconstruire le génome à partir d'innombrables petites séquences. Le diagramme qui image cette méthode est présenté à la figure 1 ci-dessous :

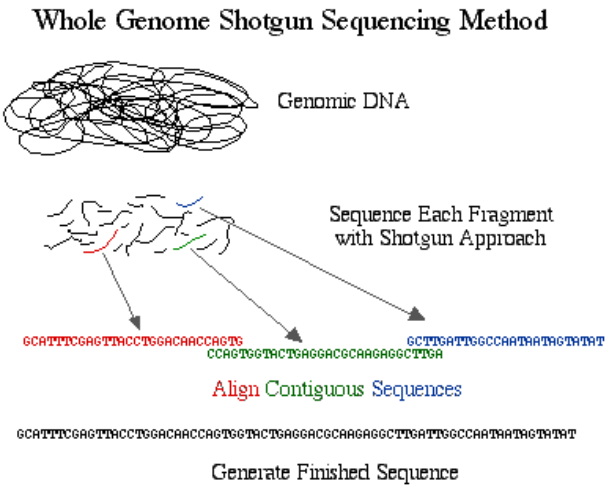

Figure 1: Diagramme de méthodologie WGS [14]

Cette méthode possède plusieurs variantes, dont l'une est le « Hierarchical Shotgun Sequencing ». Cette variante de la méthode introduit le concept des « contigs » qui servent à référencer les différents segments du génome. Ainsi, en cassant l'ADN en segments, ces derniers sont identifiés et une carte des « contigs » est produite. Le séquençage des « contigs » permet la reconstruction de la séquence initiale par alignement. Le diagramme de cette variante de la méthode est présenté à la figure 2.

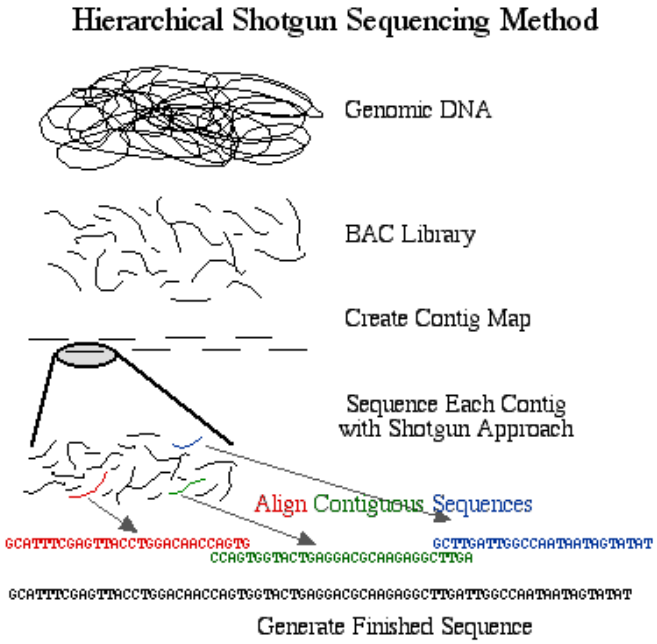

Figure 2: Diagramme de méthodologie Hierarchical Shotgun Sequencing [14]

1.3.2 Le génotypage

Le domaine de génotypage, quant à lui, vise à identifier la variation d'un gène sur le génome d'un individu ou un groupe d'individus vivants. Les applications de génotypage sont multiples. Dans le domaine médical, la compréhension de l'expression du génome en traits phénotypiques peut permettre de dépister les maladies. Dans le domaine de la sécurité, les études des composants du génome permettent l'identification des individus. En agronomie, les études génotypiques peuvent permettre d'effectuer les sélections variétales.

Plusieurs techniques de génotypage existent actuellement [15]. Notamment, l'une des techniques les plus populaires est le « Whole Genome Association Studies (WGAS) » [16]. Cette technique consiste en analyse des marqueurs dans les séquences d'ADN afin de comprendre la variation génétique associée aux maladies. La réalisation de ces études nécessite d'avoir des données pertinentes représentant un nombre suffisant des polymorphismes marqués en lien avec une maladie. Typiquement, les études WGAS sont conduites dans le cadre d'études cliniques contrôlées sur une assez grande population

d'individus atteints d'une maladie, accompagnés d'une population témoin en bonne santé qui sert de référence.

Une étude WGAS est typiquement réalisée en deux étapes. La première étape consiste en une étude clinique, nécessaire au recueil contrôlé des données. La deuxième étape se concentre sur l'analyse des données à l'aide de logiciels spécialisés. L'un de ces logiciels est le logiciel Plink [17]. Plink est un regroupement de techniques d'analyses de type WGS qui a été développé par les universités d'Harvard et de MIT. Ce logiciel libre est spécialisé dans l'analyse des données génotypiques et phénotypiques et est largement utilisé dans le domaine de génotypage actuellement.

1.3.3 Les défis dans le domaine d'analyse génétique

Bien que similaires, les domaines du séquençage et du génotypage se concentrent sur des objectifs assez différents. De ce fait, ces deux secteurs ne se développent pas à la même vitesse et de la même manière.

Le secteur de séquençage génétique est composé d'une faible quantité de grands acteurs universitaires et privés. De ce fait, ce domaine de recherche est assez bien organisé et structuré. Il implique des consortiums internationaux qui permettent une convergence vers la normalisation et la standardisation des approches, de la terminologie et des formats de fichiers. Notamment, dans le domaine du séquençage, on retrouve des formats de fichiers standards tels que SAM, BAM et VCF. Cette normalisation favorise le développement d'offres des services infonuagiques spécialisés tel que présenté au début de cette section.

Le secteur de génotypage, de son côté, offre actuellement des logiciels d'une grande variété et dans toutes sortes de domaines de la recherche. Il peut notamment s'agir du dépistage des maladies, de la sélection variétale en agronomie, de l'identification des individus et bien d'autres. De ce fait, les acteurs sont beaucoup plus nombreux, mais souvent d'importance plus réduite. En considérant uniquement ses applications médicales, ce domaine est composé

de différentes équipes de recherche qui travaillent sur la compréhension des différentes maladies mettant en œuvre des études cliniques variées, uniques et qui se compétitionnent entre elles. L'hétérogénéité de cet écosystème génère une absence de normalisation et de standardisation des données, des processus ainsi que des outils/logiciels produits.

1.4 Conclusion

Ces dernières années, plusieurs initiatives de traitements des données génétiques utilisant des technologies Big Data ont vu le jour. La plupart de ces projets ce sont intéressés au domaine de séquençage génétique. En revanche, peu d'initiatives s'intéressent au domaine du génotypage.

L'organisation structurelle de l'écosystème de séquençage génétique a permis d'aboutir à une normalisation du domaine, ce qui a favorisé l'émergence d'offres de services infonuagiques et de Big Data. À ce jour, le domaine de recherche du génotypage ne possède que peu d'activités de normalisation. Ce manque de standardisation rend plus lente et complexe l'émergence d'outils de traitements de données utilisables facilement.

CHAPITRE 2

Introduction des traitements Big Data dans le domaine de génotypage

2.1 Introduction

Nous avons vu que l'une des problématiques majeures du domaine du génotypage est son manque de normalisation. Bien que le domaine soit composé d'équipes multiples travaillant sur des problématiques variées, il s'oriente principalement sur le développement de méthodes WGAS et le logiciel libre Plink est très utilisé. Toute avancée vers une normalisation du domaine accélérerait grandement la conception et l'apparition d'une nouvelle génération d'outils plus complets, plus généralistes et surtout plus performants.

C'est dans ce contexte que ce projet de recherche tente d'introduire des technologies de traitement Big Data dans le domaine du génotypage. Ce projet se déroule en partenariat avec le CRCHUM. L'équipe partenaire permet d'accélérer la compréhension de ce domaine complexe en proposant un cas d'étude concret à base d'une étude clinique réelle.

2.2 Méthodologie appliquée

Le projet tente d'introduire une nouvelle technologie dans un domaine complexe qui est celui de génotypage. Pour le faire, il était nécessaire d'aborder directement deux axes qui sont la technologie et le métier.

2.2.1 Axe métier - domaine de génotypage

L'axe métier concerne (a) la compréhension générale du domaine de génotypage, (b) l'identification du scénario d'application et (c) la compréhension des traitements actuels.

a) La compréhension générale du domaine de génotypage

La compréhension générale du domaine de génotypage nécessite d'effectuer une étape d'autoformation. La clé d'une bonne autoformation est de bien cibler le périmètre d'apprentissage. Si le périmètre est trop vaste, cela engendre une perte de temps et l'écartement des objectifs fixés. Si le périmètre n'est pas suffisamment profond, cela présente un risque d'échec du projet lié à la non-compréhension des objectifs. Les étapes d'une autoformation sont les suivantes :

- Délimitation du périmètre
- Collecte des informations
- Lecture et mise en contexte de l'information

b) L'identification du scénario d'application

Le domaine de génotypage est très vaste, il était nécessaire de cibler un cas d'application bien spécifique. Cela a été réalisé en partenariat avec l'équipe de recherche de CRCHUM. L'identification de scénario d'application a été réalisée en plusieurs étapes :

- Échange avec les chercheurs de CRCHUM concernant leurs objectifs et leurs méthodologies de travail.
- Priorisation de différents scénarios
- Choix préliminaire du scénario
- Proposition des hypothèses sur les modélisations et les utilisations possibles.
- Ajustement des hypothèses avec les chercheurs de CRCHUM.
- Validation du scénario d'application

c) La compréhension des traitements actuels

Il est nécessaire de connaître un processus pour pouvoir proposer son amélioration. C'est une condition nécessaire mais non suffisante. La compréhension du processus de traitement existant a été réalisée en plusieurs étapes :

- Identification des outils utilisés par l'équipe de chercheurs de CRCHUM dans le scénario d'application
- Identification de l'outil central et représentatif de la complexité

- Conception de la preuve de concept permettant de valider les hypothèses définies en proposant une solution alternative au processus de traitement identifié
- Validation de la pertinence du choix de la preuve de concept
- Rétro-ingénierie de l'outil utilisé dans l'optique d'identification et de compréhension détaillée des algorithmes

2.2.2 Axe technologique - Big Data

L'axe technologique concerne (a) la compréhension des technologies Big Data et (b) la réalisation de la preuve de concept.

a) La compréhension des technologies Big Data

Afin de proposer une solution Big Data pour le domaine de génotypage, il était nécessaire de comprendre la technologie Big Data mais aussi de se renseigner sur ce qui ce fait actuellement dans le monde. Les étapes sont les suivantes :

- Recherche bibliographique sur les applications Big Data en génétique
- Identification de solutions les plus pertinentes pour le cadre d'application
- Autoformation sur les différentes solutions identifiées

b) La réalisation de la preuve de concept

La réalisation de la preuve de concept consiste en ingénierie et développement du logiciel. L'outil réalisé implémente les différentes technologies Big Data ainsi que les algorithmes de génomique.

2.3 Cas d'application (cas d'étude)

L'équipe du CRCHUM travaille sur un cas spécifique de génotypage visant à comprendre la maladie du diabète. Pour ce faire, l'équipe a effectué une étude clinique sur des personnes atteintes de diabète collectant ainsi de l'information génotypique et phénotypique des

individus sur plusieurs années de la maladie et des traitements. L'étude clinique étant terminée, l'équipe de chercheurs se concentrent à l'analyse des données collectées visant à comprendre l'impact des génotypes sur les traits phénotypiques. Afin d'effectuer ces analyses, l'équipe de bio-informaticiens du CRCHUM utilise le logiciel libre Plink ainsi que des algorithmes spécifiques additionnels pour des besoins d'analyses plus poussées.

Les chercheurs ont partagé les principaux points d'amélioration souhaitables :

- Améliorer la normalisation des données et des processeurs;
- Améliorer la traçabilité des traitements;
- Garantir la capacité de traitement avec la croissance des données.

2.4 Les enjeux et les difficultés

Comme nous l'avons observé, l'enjeu majeur du domaine du génotypage est le manque de normalisation à tous les niveaux. Afin d'adresser cette problématique, une preuve de concept a été conçue afin de démontrer la possibilité d'améliorer la situation. Ce logiciel expérimental, qui est une preuve de concept, implémenterait une partie des fonctionnalités de Plink mais en utilisant des technologies Big Data. Ce logiciel expérimental, surnommé adam-ibs proposerait une approche permettant la réutilisation et l'extension du modèle de données actuellement disponible et proposé par le projet ADAM de l'Université Berkeley.

La preuve de concept doit satisfaire deux contraintes supplémentaires qui sont (a) la mise à l'échelle et (b) la traçabilité des opérations.

- Pour assurer la mise à l'échelle de l'outil, soit de garantir que les capacités de traitement seront facilement adaptables lors de la croissante des données, la preuve de concept utilisera la technologie Spark. Cette dernière est une technologie Big Data permettant d'effectuer le traitement des données en mémoire d'une manière répartie sur une grappe des serveurs.
- La traçabilité des analyses est nécessaire lors de la reproduction d'une découverte par les chercheurs. Pour faciliter la mise en place de la traçabilité des données et des

traitements, la preuve de concept implémentera une solution de stockage de données immuable. Il s'agit notamment des solutions Apache Avro et Parquet.

2.5 Réalisation de la preuve de concept

Parmi les initiatives citées précédemment, le projet de logiciel libre ADAM, réalisé par les chercheurs de l'université Berkeley est appropriée pour appuyer la démarche de ce projet de recherche. Le projet ADAM se concentre sur les besoins en séquençage en proposant une plateforme de traitement des données Big Data en utilisant les technologies Hadoop, Spark, Avro et Parquet. Le caractère ouvert du projet ADAM permet d'utiliser des modèles et concepts développés existants et de les enrichir avec les besoins du domaine du génotypage. De même, l'enrichissement d'ADAM par les résultats de cette recherche permet un retour à la communauté. La coopération et l'innovation ouverte permettent d'apporter une meilleure viabilité et visibilité du projet.

2.5.1 Objectifs et exigences

Cette preuve de concept consiste à migrer une partie des fonctionnalités de Plink sur la technologie Big Data utilisant Spark pour le traitement reparti ainsi que Avro et Parquet pour la persistance de données sur disque. Également, le projet doit s'appuyer sur les concepts et modèles proposés par le projet ADAM.

L'un des processus centraux de Plink régulièrement utilisé dans les études WGAS est nommé IBS-MDS. Ce dernier est composé d'un ensemble des traitements de données successives. La preuve de concept se concentre sur l'adaptation du processus IBS-MDS au modèle d'ADAM. Puis, sur l'implémentation d'une partie du processus IBS-MDS en langage Scala, pour être exécutée sur une infrastructure Big Data à l'aide de la technologie émergente Spark.

La réalisation de cette preuve de concept nécessite une maîtrise complète de la technologie Big Data ciblée, ainsi que la compréhension fonctionnelle et technique de Plink et d'ADAM. L'outil Plink est accompagné d'une documentation complète d'utilisation présentant les

différentes fonctionnalités. Cependant, la compréhension complète des algorithmes implémentés dans Plink nécessite des efforts en rétro-ingénierie afin d'identifier et d'extraire les algorithmes concernés. La compréhension fonctionnelle de ce module de Plink, quant à elle, nécessite une compréhension métier du domaine de la génétique.

2.5.2 L'organisation de l'outil adam-ibs

Le logiciel adam-ibs résultant sera principalement développé en langage Scala à la version 2.10.0. Toutefois, de nombreuses dépendances Java y sont présentes. Le JDK 1.7 de Java sera utilisé pour satisfaire la compatibilité avec Spark, Avro et Parquet. Afin de faciliter la gestion des dépendances, le projet implémentera le logiciel Maven. Ce dernier facilite la gestion des dépendances ainsi que des versions de logiciels.

Le logiciel Maven permet d'effectuer la gestion des dépendances de l'outil adam-ibs. Par l'intermédiaire de son fichier de configuration pom.xml, Maven permet de structurer l'ensemble du code. Notamment, l'outil permet de définir les langages et les versions utilisés, de définir les différents modules du projet, d'inclure l'ensemble des dépendances logicielles, de définir les étapes et l'ordre de compilation, etc.

Ainsi, la présentation de la structure des fichiers de configuration de Maven permet de présenter l'organisation de l'outil d'adam-ibs.

Au niveau de structure du projet, adam-ibs comporte deux modules adam-ibs-core et adam-ibs-data :

```
  <groupId>com.ets.mgl804</groupId>
  <artifactId>adam-ibs</artifactId>
  <packaging>pom</packaging>
  <version>0.1.0</version>
  <name>Adam IBS</name>

  <modules>
    <module>adam-ibs-core</module>
    <module>adam-ibs-data</module>
  </modules>
```

Le module adam-ibs-data permet de définir et de générer le modèle de données qui est utilisé par le module adam-ibs-core.

La compilation du module adam-ibs-data génère un fichier « .jar » qui intègre le modèle de données utilisé :

```
    <artifactId>adam-ibs-data</artifactId>
    <name>Adam IBS Data</name>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.avro</groupId>
                <artifactId>avro-maven-plugin</artifactId>
                <version>1.7.7</version>
                <executions>
                    <execution>
                        <id>schemas</id>
                        <phase>generate-sources</phase>
                        <goals>
                            <goal>schema</goal>
                            <goal>protocol</goal>
                            <goal>idl-protocol</goal>
                        </goals>
                        <configuration>

<sourceDirectory>${project.basedir}/src/main/resources/com/ets/mgl804/avro
</sourceDirectory>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
```

Une fois que « adam-ibs-data-0.1.0.jar » est compilé, il est utilisé (c.-à-d. en tant que dépendance) dans le module adam-ibs-core :

```
        <!-- Data -->
        <dependency>
            <groupId>com.ets.mgl804</groupId>
            <artifactId>adam-ibs-data</artifactId>
            <version>0.1.0</version>
        </dependency>
```

La compilation de l'ensemble du projet adam-ibs compile les modules selon leur ordre de dépendances et génère deux fichiers « .jar ». Un premier fichier « .jar » comporte le code

spécifique au projet, l'autre fichier « .jar » comporte l'ensemble des dépendances et identifie la classe principale. Il peut être exécuté directement sur une grappe d'ordinateurs opérant Spark :

```
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-assembly-plugin</artifactId>
                <configuration>
                    <archive>
                        <manifest>
                            <addClasspath>true</addClasspath>

<mainClass>com.ets.mgl804.core.Main</mainClass>
                        </manifest>
                    </archive>
                    <descriptorRefs>
                        <descriptorRef>jar-with-
dependencies</descriptorRef>
                    </descriptorRefs>
                </configuration>
                <executions>
                    <execution>
                        <phase>package</phase>
                        <goals>
                            <goal>single</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
```

2.5.3 La technologie émergente Big Data : Spark

Apache Spark est une technologie émergente du domaine du Big Data créée par l'université Berkeley. Elle offre un moteur de traitement réparties de données à grande échelle. Spark est compatible avec Hadoop et permet de répartir le traitement des données sur des grappes d'ordinateurs facilement. Ce cadriciel Big Data permet d'effectuer aussi bien les traitements en lots que des requêtes interactives sur des quantités massives de données.

Spark peut être déployé localement ou sur une grappe distribuée d'ordinateurs. La configuration locale constitue une seule instance Spark sur un ordinateur local ou personnel. Cette configuration est notamment utile lors du développement et des tests d'exécution d'une solution logicielle. La configuration distribuée vise le déploiement d'une solution logicielle

Spark en multiples instances travaillant d'une manière répartie sur une grappe d'ordinateurs afin d,en tirer la puissance de calcul.

Une configuration « Cluster » de Spark consiste en un déploiement d'un nœud maître - nommé « Driver Node », et d'un ensemble des nœuds esclaves - appelés « Worker Nodes ». Le nœud maître est responsable de la gestion de la grappe d'ordinateurs dans sa globalité ainsi que de la bonne exécution des applications en mode distribué.

Une application exécutable sur une grappe Spark peut être écrite en Java, en Python ou en langage de programmation Scala. L'appellation Scala provient de la concaténation de « sca » pour « scalable » et de « la » pour « language ». Cela signifie littéralement « langage pouvant évoluer (c.-à-d. élasticité) et se mettre à l'échelle », Scala est conçu pour les applications Big Data. De plus, Scala est utilisé par le projet ADAM de Berkeley. En prenant en considération tous ces éléments, c'est bien le langage Scala qui est adapté à notre situation et qui sera utilisé dans ce projet de recherche appliquée. Le déploiement et l'exécution d'une application sur une grappe Spark sont effectués en définissant un « SparkContext » qui coordonne l'ensemble ces deux processus, voir la figure ci-dessous.

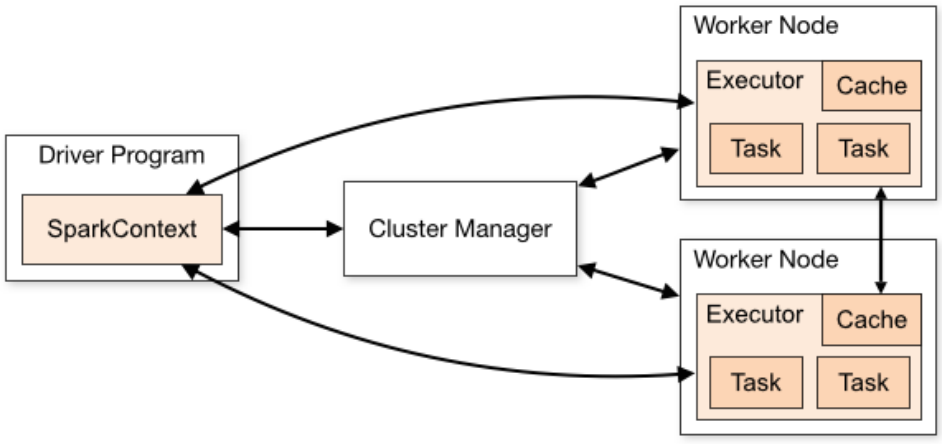

Figure 3: Composantes de l'environnement Spark en mode "Cluster" [18]

Au moment du démarrage de ce projet de recherche, la version disponible de Spark était la version 1.4.1. Cette dépendance est ajoutée au fichier Maven pom.xml du projet :

```
<spark.version>1.4.1</spark.version>
...
<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-core_2.10</artifactId>
  <version>${spark.version}</version>
</dependency>
```

Par la suite il est nécessaire de définir un objet « AppContext » dans lequel est défini le «SparkContext» :

```
object AppContext {
  val conf = new SparkConf().setAppName("Adam-IBS").setMaster("local")
  val sc = new SparkContext(conf)
  var logger = LoggerFactory.getLogger(this.getClass)
  logger.info("Load AppContext")
}
```

Cette configuration établit que l'application logicielle sera nommée « Adam-IBS » et que par défaut elle sera exécutée sur Spark en mode local utilisant une seule unité de traitement (c.-à-d. mon ordinateur personnel). Cette application pourra facilement être exécutée sur tout autre environnement Spark (c.-à-d. une grappe de calcul plus puissante) en ajoutant seulement l'adresse IP (ou le nom d'hôte) du nœud maître de la grappe Spark concernée. Cette approche novatrice à l'élasticité rend facile la mise à l'échelle une fois notre logiciel conçu.

2.5.4 Technologies Big Data : Avro et Parquet

Les technologies « Apache Avro » et « Apache Parquet » permettent d'effectuer la persistance des données sur disque. « Avro » effectue les étapes de sérialisation/désérialisation, et « Parquet » effectue les étapes d'écriture/lecture sur disque.

Les deux dépendances à ces technologies sont alors ajoutées au projet :

```
<!-- Avro -->
```

```
<dependency>
    <groupId>org.apache.avro</groupId>
    <artifactId>avro</artifactId>
    <version>${avro.version}</version>
</dependency>

<!-- Parquet -->
<dependency>
    <groupId>com.twitter</groupId>
    <artifactId>parquet-avro</artifactId>
    <version>${parquet.version}</version>
</dependency>
```

« Avro » est un système de sérialisation des données. Contrairement à d'autres outils de sérialisation, « Avro » s'appuie sur le concept de schéma de données. Le schéma de données est défini en langage de programme « JSON » qui est transmis avec les données. Ce principe rend la sérialisation rapide et légère et permet d'effectuer la reconstruction dynamique des données facilement. Ainsi, il est possible de reconstruire les données sans initialement en connaître leur structure interne.

« Parquet » de son coté, est un format de fichier à stockage en colonnes efficace pour des applications transactionnelles (c.-à-d. de style OLAP). Cette technologie Big Data est compatible avec l'écosystème Hadoop ainsi qu'avec la technologie de sérialisation « Avro » [19]. L'annexe I présente comment effectuer l'intégration de la technologie « Parquet » avec la technologie « Avro » et les autres éléments de l'écosystème de Hadoop.

Grâce à la compatibilité de « Parquet », l'écriture des objets « Avro » dans « Parquet » s'effectue naturellement en suivant les étapes de la figure 4 [16].

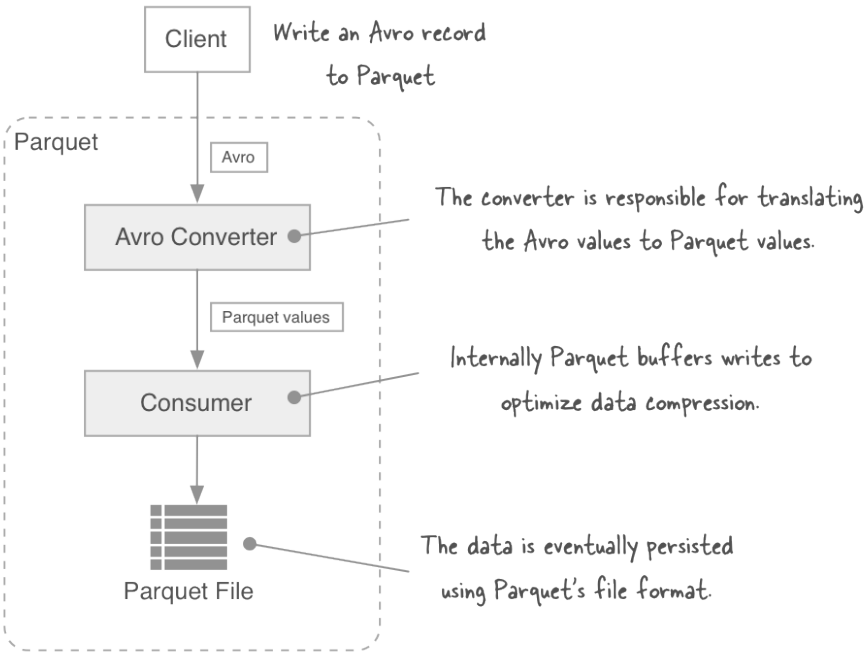

Figure 4: Écriture d'objets Avro dans Parquet [16]

L'implémentation de la persistance est effectuée à l'aide de l'écriture de classes génériques. Ces dernières permettent d'effectuer l'écriture dans un fichier « Parquet » ainsi que la lecture depuis un fichier « Parquet ». Lors de la lecture, le schéma de données est récupéré et les données sont reconstruites.

Sans entrer dans les détails techniques de chaque méthode, la classe « ParquetWiter[T] » permet de persister sur le disque un tampon des éléments de la classe « T » suivant un schéma défini.

```
class ParquetWriter[T](data:scala.collection.mutable.Buffer[T], filename:
String, avroSchemaInput:Schema) {
  private val logger = LoggerFactory.getLogger(this.getClass)
  private val DATA_PATH = "DATA/avro/"
  private val fileName = filename
  private val avroSchema = avroSchemaInput
  private val conf = AppContext.conf
  private val sc = AppContext.sc
  private val sqc = new SQLContext(sc)
```

```
  private val dataToPersist = data

  def persistData() {...}
  def writeToFile(parquetFilePath:Path) {...}
  def initialiseParqurFile() : Path= {...}
  def deleteIfExist(fileName:String) {...}
}
```

La classe « ParquetLoader[T] » permet de reconstituer les données contenues dans un fichier « Parquet ». Également, cette classe permet aussi de visualiser le schéma de données contenu dans le fichier « Parquet » ainsi que d'afficher quelques éléments de ses données.

```
class ParquetLoader[T](filename:String) {
  private val logger = LoggerFactory.getLogger(this.getClass)
  private val fileName = filename
  private val conf = AppContext.conf
  private val sc = AppContext.sc
  private val sqc = new SQLContext(sc)
  sqc.sql("SET spark.sql.parquet.binaryAsString=true")
  private val parquetFilePath:Path = new Path(this.fileName)
  private var loadedData = scala.collection.mutable.Buffer[T]()

def showContent(): Unit = {...}
def showSchema() {...}
def loadData() : scala.collection.mutable.Buffer[T] = {...}
def viewLoadedContent() {...}
}
```

Par la suite, la persistance des donnés dans les fichiers « Parquet » s'effectue à l'intérieur d'une méthode en instanciant la classe « ParquetWriter[T] », tel que démontré ci-dessous :

```
...
private var listPairwiseIbaIbd = scala.collection.mutable.Buffer[PairwiseIbsIbd]()
...
  def persistData(filename:String) {
    val parquetWriter = new ParquetWriter[PairwiseIbsIbd](this.listPairwiseIbaIbd,
                            filename, PairwiseIbsIbd.getClassSchema)
    parquetWriter.persistData()
```

```
  }
...
```

De même, la lecture des fichiers « Parquet » s'effectue en instanciant la classe « ParquetLoader[T] » :

```
...
private val listIndividuals = new ParquetLoader(filename)
...
val listIndividuals = this.listIndividuals.loadData()
...
```

Ainsi, avec ce type de conception d'implémentation, la persistance et la reconstitution des données à destination et provenant de fichiers « Parquet » s'effectue d'une manière générique, simple et facilement maintenable.

2.5.5 Le modèle de données ADAM

Le projet ADAM offre une solution logicielle ouverte (c.-à-d. en logiciel libre) pour le traitement Big Data des données génétiques pour le domaine de séquençage du génome. Notamment, ce projet propose un modèle de données qui répond aux besoins de séquençage et qui peut être adapté afin de prendre en compte les besoins de génotypage du laboratoire ou la preuve de concept sera réalisée. Il s'agit d'un modèle de données intitulé « bdg-format » [20] présenté en Annexe II. Cette preuve de concept s'appuiera sur le modèle de données proposé par ADAM et l'enrichira avec les besoins de génotypage. Le modèle répondant suffisamment aux besoins de stockage des données génotypique, le travail consistera majoritairement à intégrer les concepts d'individus en associant à la fois les données génotypiques avec les données phénotypiques.

Le projet « bgd-formats » permet déjà de générer le code Java structurant le modèle de données. Le schéma de données de ce dernier est défini en format JSON. Le code source sera importé dans le projet de preuve de concept adam-ibs et constituera le nouveau module adam-ibs-data. Le nouveau modèle de données sera complété en décrivant un nouveau

schéma aussi en format JSON. La compilation de ce module permettra de produire un fichier « .jar » qui est utilisé dans le reste du projet (c.-à-d. adam-ibs-data) en tant que dépendance.

2.5.6 Introduction à l'outil WGAS Plink [21]

Plink est un outil de « Whole Genome Association Studies » (WGAS) développé en langage C/C++ par les universités Harvard et MIT. C'est un outil très populaire et largement utilisé par les bio informaticiens au travers le monde actuellement. Plink fonctionne en mode « ligne de commande » ce qui signifie qu'il n'y a pas d'interface utilisateur simple d'utilisation. À chaque commande de traitement de l'information, Plink effectue le traitement demandé sur (a) des fichiers en entrée et (b) produit des fichiers en sortie. De manière générale, les fichiers en entrée et en sortie possèdent des formats différents. L'exécution d'un processus complet de traitement des données passe donc par l'exécution successive des commandes en générant une multitude de fichiers intermédiaires.

2.5.7 Le processus IBS-MDS

Plink propose un large spectre des fonctionnalités réparties sur cinq domaines : a) gestion des données, b) statistiques sommaires, c) stratification des populations, d) analyses associatives et e) estimation IBD [17]. L'un des objectifs principaux du génotypage est la compréhension de l'impact du génome des individus sur leur expression phénotypique. Pour se faire, les bio-informaticiens utilisent les méthodes de stratification des populations afin d'isoler les caractéristiques intéressantes dans le cadre de leurs études. Ainsi, dans un premier temps, le projet s'intéresse plus spécifiquement aux fonctionnalités de stratification des populations de Plink et plus précisément le processus central qui est le processus IBS-MDS. Ce processus est représentatif de la complexité globale de l'ensemble d'autres fonctionnalités qui pourront être implémentées par la suite.

Ce processus concerne plusieurs traitements successifs sur les données tel que présenté à la figure 5.

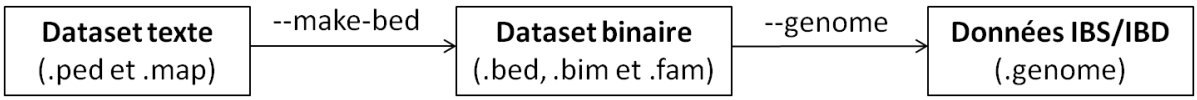

Figure 5: Vue de l'ensemble de processus de calculs IBS-MDS de Plink

Extension des fichiers représentant les formats de données concernés par le processus :

.ped - Le fichier original contenant l'information génotypique.

.map - Le fichier contenant l'information sur les variants qui accompagne le .ped.

.bed - Le fichier binaire contenant les informations de génotypage et de variantes. Doit être accompagné de fichiers .bim et .fam.

.bim - Le fichier contenant l'information sur les variants qui accompagne le .bed.

.fam - Le fichier d'information qui accompagne le .bed.

.genome - Le fichier issu de la fonction --genome.

Fonctions concernées par le processus :

--make-bed - L'opération qui produit un ensemble des fichiers binaires (.bed, .bim, .fam) à partir des fichiers sources (.ped et .map).

--genome - L'opération qui effectue les calculs IBS-MDS sur les données sources peut être exécuté avec l'option "full" introduisant des calculs supplémentaires. L'opération est exécutée sur les fichiers sources (texte ou binaires) et produit un fichier « .genome ».

--file L'option précisant les fichiers texte en entrées des opérations.

--bfile L'option précisant les fichiers binaires en entrées des opérations.

--out L'option précisant le nom du fichier sortant de l'opération.

Une fois que le contexte et l'exemple d'une fonction Plink à convertir vers ADAM sont identifiés, le travail de rétro-ingénierie consiste à comprendre les formats de données et les algorithmes de traitement effectués par IBS-MDS. Le détail de cette étude ainsi que les formats de données utilisés par le processus IBS-MDS, de Plink, est présenté en annexe III. La section suivante discute de l'intégration des besoins de ce module Plink dans le modèle de données proposées actuellement par ADAM et qui devra être adapté.

2.5.8 Intégration des modèles de données

Nous avons vu que le modèle de données d'ADAM répond aux besoins de séquençage génétique et de ce fait il comporte uniquement les données génotypiques. Plus généralement le domaine de génotypage et plus spécifiquement l'outil Plink, se concentrent sur la notion de l'individu. L'individu est porteur d'éléments génotypiques et phénotypiques indispensables.

Ainsi, pour répondre aux besoins de l'outil Plink tout en complétant le modèle d'ADAM, il a été décidé de centrer le nouveau modèle sur l'individu. L'individu devient l'élément central du modèle futur. Ce dernier comporte les éléments permettant de retracer ses liens de parenté avec d'autres individus (c.-à-d. individu paternel, maternel, famille...) et comporte son expression génotypique et son expression phénotypique. L'expression génotypique de l'individu est actuellement couverte par le modèle ADAM publié. Initialement, la modélisation de l'expression phénotypique de l'individu couvre les besoins de Plink dans le contexte du processus IBS-MDS. Par contre, l'élicitation des besoins des chercheurs du laboratoire du Dr Hamet, en génotypage (voir la session 3.3.1), nécessitera d'adapter cette modélisation.

L'annexe IV présente, d'une manière plus détaillée, les extensions apportées au modèle ADAM existant. Également, l'annexe précise comment est effectuée la mise en correspondance des données depuis les formats actuels de Plink.

2.5.9 Réingénierie des processus

La figure 6 présente une vue d'ensemble des processus des calculs IBS-MDS de Plink. Afin de produire le fichier « .genome », le flux de travail comporte deux étapes de transformation sur les données sources « .ped » et « .map ». La première étape est effectuée par l'opération « --make-bed » et la seconde par l'opération « --genome ».

L'opération « --make-bed » a un objectif double. Le premier objectif est de produire un fichier binaire à base des informations contenues dans les données sources. Ce fichier binaire permet d'optimiser les traitements postérieurs tels que les traitements effectués par « --genome ». Le second objectif est d'enrichir les données, la fonction effectue un calcul des métadonnées statistiques qui complètent les données d'origine. L'opération « --genome » procède à l'analyse statistique des données enrichies par « --make-bed ».

La solution technologique proposée par ce projet, adam-ibs, n'aura plus besoin d'étapes de prétraitement des données. Au niveau de typologie des fonctionnalités, on parlera de « l'intégration des données », de « l'enrichissement des données » ainsi que de « l'exploitation des données ». Appliquée au contexte du fonctionnement de Plink, la typologie des trois processus est présentée ci-dessous :

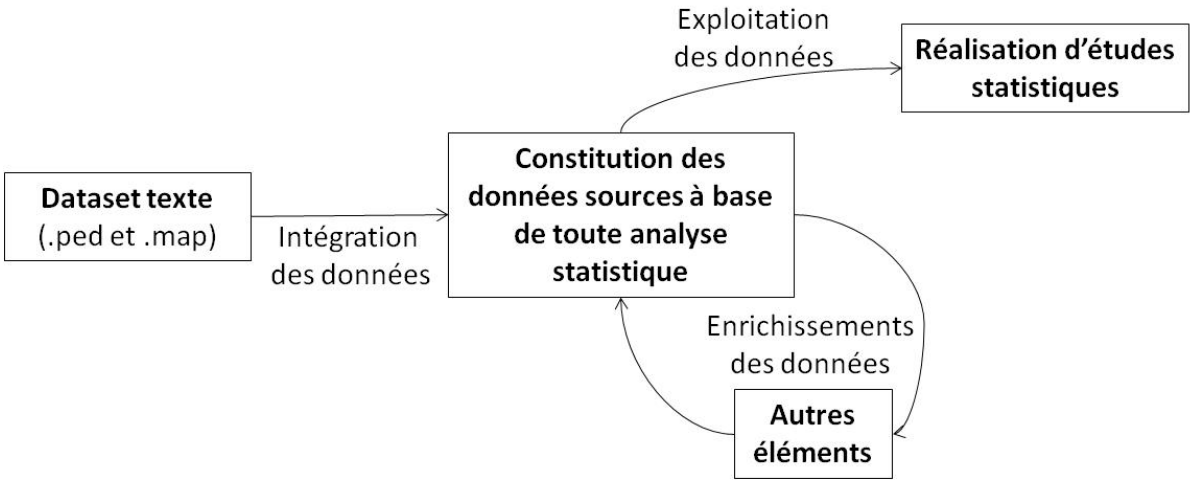

Figure 6: La typologie des processus proposés pas adam-ibs

Appliquée au périmètre du processus IBS-MDS de Plink, « l'intégration des données » correspond à l'intégration des données contenues dans ".ped" et ".map" dans le modèle adam-ibs. L'étape de « l'enrichissement des données » correspond au calcul des métadonnées effectué par « --make-bed ». L'étape de « l'exploitation des données » correspond à la fonction « --genome » qui produit un rapport d'analyses statistiques.

Dans certains cas, pour simplifier les traitements, les processus génériques décrits précédemment peuvent être optimisés. C'est notamment le cas d'application de la fonction « --make-bed » qui présente un caractère d'intégration et d'enrichissement des données. Ces deux opérations peuvent être réalisées en une seule manipulation dans l'adam-ibs :

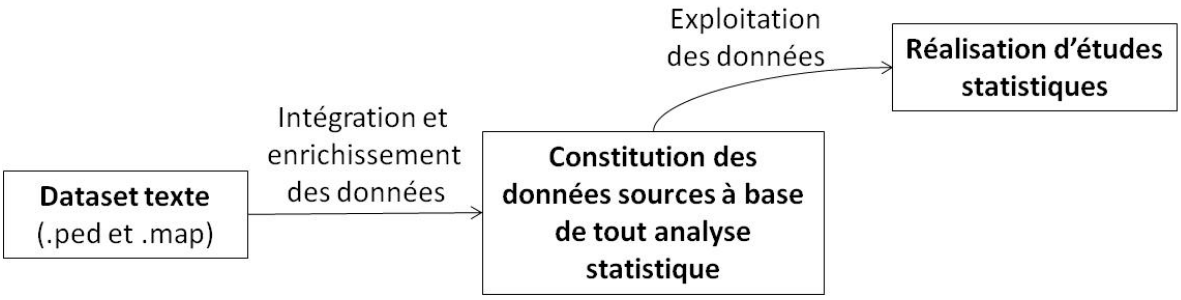

Figure 7: Optimisation du processus « --make-bed »

De ce fait, adam-ibs implémentera deux fonctions :

--make-bed : Calcul des métadonnées sur les données sources et l'intégration de l'ensemble dans le modèle de données adam-ibs.

--genome : Exploite les données originelles modélisés (un fichier parquet respectant le modèle adam-ibs) pour générer un rapport d'analyses statistiques autoporteur (contenant les données originelles associées à l'analyse) centré sur la classe « Pairwise_IBS_IBD » (se référer à l'Annexe IV).

2.5.10 Rétro-ingénierie et implémentation des algorithmes

Le modèle et les processus étant définis, il est nécessaire à cette étape de procéder au ciblage des algorithmes à étudier et convertir. Ensuite, il sera nécessaire de procéder à la rétro-ingénierie des algorithmes et de leur ré implémentation dans le nouveau contexte et modèle d'adam-ibs.

La première fonctionnalité implémentée est « --make-bed ». Cette dernière importe les données issues des fichiers « .ped » et « .map » et effectue un premier calcul des métadonnées.

L'implémentation s'effectue en trois étapes :

1. Extraction des données contenues dans « .ped » et « .map » tout en vérifiant leur cohérence. Les données sont mises en correspondance dans le modèle adam-ibs telles que présentées dans la partie 2.3.7;
2. Transformation des données et leur enrichissant avec les métadonnées apportées par « --make-bed » de Plink. L'analyse des lots des données « .ped + .map » et « .bed + .bim + .fam » montre que la fonctionnalité de Plink « --make-bed » apporte les métadonnées « Reference Allele (A1) » et « Alternate Allele (A2) » contenues dans le fichier « .bim ». La documentation et les tests démontrent que ces valeurs sont issues du calcul statistique sur les données originelles. A1 apparaît alors en tant que l'allèle minore et A2 l'allèle majeur du lot de données présent;
3. Persistance des données sur disque est effectué tel que présenté dans la partie 2.3.4.

La seconde fonctionnalité implémentée est « --genome ». Cette dernière réalise les calculs IBS/IBD et génère un ficher « .parquet » contenant une liste des classes « Pairwise_IBS_IBD » présentée en annexe IV. Pour le faire, un travail de rétro-ingénierie a été réalisé sur le code source de Plink ce qui a permis l'extraction des algorithmes associés. Les algorithmes de la fonction sont décrits plus en détail sur la page wiki du projet adam-ibs [22].

2.5.11 Autres considérations

Outre la réalisation des fonctionnalités métiers décrites précédemment, adam-ibs implémente des fonctionnalités de support nécessaires à son fonctionnement. Notamment, le projet

implémente une solution de gestion des paramètres d'exécution en ligne de commande ainsi qu'une solution de centralisation et de gestion des logs.

2.5.11.1 Intégration du parseur de ligne de commande

Afin de simplifier l'exécution des différentes fonctionnalités d'adam-ibs, ce dernier implémente une solution existante de parseur de ligne de commande. Après l'analyse comparative de plusieurs alternatives possibles, le parseur scallop a été implémenté. Son intégration s'effectue en deux étapes. La première étape concerne l'ajout de la dépendance Maven, la deuxième étape concerne le paramétrage de l'outil.

```
<!-- scallop for CLI parser -->
<dependency>
    <groupId>org.rogach</groupId>
    <artifactId>scallop_2.10</artifactId>
    <version>0.9.5</version>
</dependency>
```

2.5.11.2 Intégration de système de gestion des logs

Le projet adam-ibs intègre une multitude des dépendances logicielles nécessaires à son utilisation (Spark, Avro, Parquet...). Chacun de ces projets intègre son propre système de gestion des logs. Afin de pouvoir les gérer efficacement, adam-ibs intègre un système de gestion des logs centralisé. Après plusieurs considérations, le choix s'est porté sur le projet Logback [23]. Ce dernier est compatible avec scala et permet l'interfaçage avec les différentes solutions alternatives existantes. La figure 8 détaille la manière dont est faite l'interconnexion avec les autres solutions de gestion des logs [20].

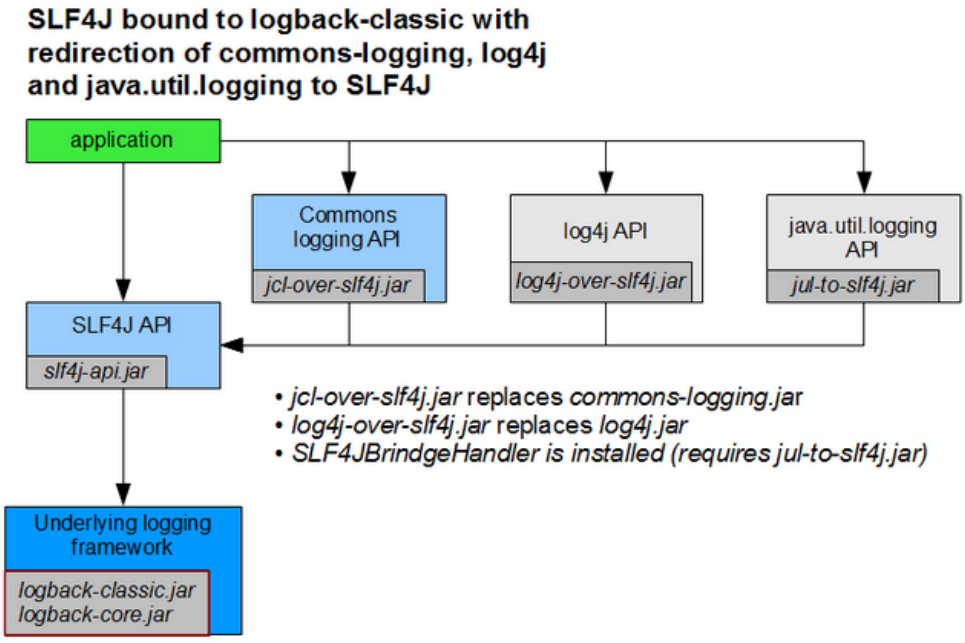

Figure 8: Interfaçage de logback [20]

Afin de faciliter la configuration de logback, adam-ibs implémente Groovy. Cette solution permet de configurer logback avec un langage dédié (DSL) au lieu d'utiliser le format de configuration basé sur XML. Cette approche facilite la configuration et la gestion de l'outil.

2.6 Conclusion

Le domaine de génotypage souffre d'un manque de normalisation des données et des processus. Ceci est principalement lié à l'écosystème du domaine ainsi qu'à la complexité de ce dernier. Une multitude des partenaires à travers le monde travaillent sur des sujets divers et exploitent à la fois les données génotypiques et phénotypiques.

Le projet adam-ibs propose une nouvelle approche permettant de modéliser les données de génotypage tout en standardisant le processus de traitement. Cette preuve de concept propose une implémentation des traitements effectués par l'outil WGAS Plink sur une technologie Big Bata. En plus de proposer un modèle générique viable, adam-ibs, cette approche permet de réduire le nombre des formats de fichier impliqués simplifiant ainsi la conception et permettant sa mise à l’échelle dans le futur.

CHAPITRE 3

Résultats et observations

3.1 Introduction

La réalisation de ce projet de maîtrise appliquée s'est effectuée en trois étapes. La première étape concerne la compréhension du domaine d'application qui est l'analyse génétique. La seconde étape concerne la rétro-ingénierie de l'outil Plink. Et la troisième et dernière étape concerne la proposition et implémentation des nouveaux modèles et processus.

Le travail réalisé a permis d'aboutir à l'implémentation d'outil Big Data nommé adam-ibs. Ce dernier fournit une preuve de concept du modèle des données, des processus, de la technologie et de la méthodologie pouvant être introduits dans le domaine de génotypage. Ce chapitre présente les fonctionnalités implémentées, l'utilisation de l'outil adam-ibs ainsi que le potentiel de son utilisation et les évolutions possibles.

3.2 Utilisation adam-ibs

Adam-ibs est une application développée sur la technologie Spark. La compilation du projet permet de produire un fichier « .jar » pouvant être exécuté sur une grappe Spark (version 1.4.1 ou ultérieure). Son exécution est effectuée à l'aide de l'utilitaire « spark-submit » contenu dans le répertoire bin d'une distribution Spark. La description d'utilisation d'adam-ibs s'effectue dans les conditions réelles utilisant un jeu de test des fichiers « .map » et « .ped » cohérents.

3.2.1 Considérations initiales

Ce test s'effectue sur une distribution de Spark v1.4.1. Dans le répertoire bin, un dossier DATA est créé contenant le fichier adam-ibs-core-0.1.0-jar-with-dependencies.jar issue de la compilation du projet ainsi que le jeu de donnés initial test.map et test.ped :

```
ikizema@Ivan-PC-Ubuntu:/opt/spark-1.4.1/bin/DATA$ ll
drwxrwxr-x 2 ikizema ikizema      4096 Nov  4 17:17 ./
drwxr-xr-x 3 ikizema ikizema      4096 Nov  4 17:14 ../
-rw-rw-r-- 1 ikizema ikizema 102318449 Nov  4 17:11 adam-ibs-core-0.1.0-jar-with-
dependencies.jar
-rw-rw-r-- 1 ikizema ikizema        22 Nov  4 17:17 test.map
-rw-rw-r-- 1 ikizema ikizema       137 Nov  4 16:56 test.ped
```

3.2.2 Exécution d'adam-ibs

Afin de voir les différentes fonctionnalités proposées par l'outil, il est préférable d'exécuter adam-ibs avec le paramètre « -h » :

```
ikizema@Ivan-PC-Ubuntu:/opt/spark-1.4.1/bin/DATA$  ../spark-submit  ./adam-ibs-core-
0.1.0-jar-with-dependencies.jar -h
      --file  <name>   Specify .ped + .map filename prefix
      --genome         Calculate IBS distances between all individuals [needs
                       --file and --out]
      --make-bed       Create a new binary fileset. Specify .ped and .map files
                       [needs --file and --out]
      --out  <name>    Specify the output filename
      --show-parquet   Show schema and data sample stored in a parquet file
                       [needs --file]
  -h, --help  <arg>    Show help message
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
15/11/04 17:17:57 INFO Utils: Shutdown hook called
```

La fonction « --make-bed » permet de générer un fichier « Parquet » en intégrant les donnés transformés contenus dans les fichiers sources. L'utilisation de cette fonctionnalité nécessite de définir le fichier d'entée (--file) ainsi que le nom de fichier de sortie (--out). Un exemple de requête d'exécution est présenté ci-dessous :

```
ikizema@Ivan-PC-Ubuntu:/opt/spark-1.4.1/bin/DATA$  ../spark-submit  ./adam-ibs-core-
0.1.0-jar-with-dependencies.jar  --file  test  --out  testMakeBed  --make-bed  >
process.MakeBed.log
```

La fonction « --show-parquet » permet de visualiser le schéma des données contenu dans le fichier « Parquet » ainsi que les premiers enregistrements dans ce dernier. « --show-parquet »

s'applique à un fichier parquet défini avec « --file ». La visualisation du fichier « Parquet » généré par « --make-bed » s'effectue de la manière suivante :

```
ikizema@Ivan-PC-Ubuntu:/opt/spark-1.4.1/bin/DATA$ ../spark-submit ./adam-ibs-core-0.1.0-jar-with-dependencies.jar --file testMakeBed.makeBed.parquet --show-parquet > show.testMakeBed.log
```

La fonction « --genome » permet de générer un fichier parquet en effectuant les analyses des données contenues dans le fichier « Parquet » généré précédemment par « --make-bed ». L'utilisation de cette fonctionnalité nécessite de définir le fichier d'entrée (--file) ainsi que le nom de fichier de sortie (--out). Un exemple de requête d'exécution est présenté ci-dessous :

```
ikizema@Ivan-PC-Ubuntu:/opt/spark-1.4.1/bin/DATA$ ../spark-submit ./adam-ibs-core-0.1.0-jar-with-dependencies.jar --file testMakeBed.makeBed.parquet --out testGenome --genome > process.Genome.log
```

De même, le fichier parquet ainsi généré peut être visualisé avec « --show-parquet » :

```
ikizema@Ivan-PC-Ubuntu:/opt/spark-1.4.1/bin/DATA$ ../spark-submit ./adam-ibs-core-0.1.0-jar-with-dependencies.jar --file testGenome.genome.parquet --show-parquet > show.testGenome.log
```

Les données originelles ainsi que le résultat d'exécution des commandes décrites précédemment sont contenus dans l'annexe V. Les tests démontrent l'utilisation du modèle présenté en annexe IV.

3.3 Évolution d'adam-ibs

L'implémentation et l'utilisation d'adam-ibs permettent d'appuyer l'idée de pouvoir proposer un modèle de données pouvant couvrir les besoins du domaine de génotypage dans sa globalité. La méthodologie permettant d'implémenter le processus IBS-MDS de Plink peut être réappliquée pour l'implémentation d'un autre processus.

Ainsi, l'évolution d'adam-ibs est possible dans deux directions principales. La première direction est axée sur la modélisation des données. Le modèle des données proposé peut être

réutilisé et complété pour répondre au maximum des exigences du domaine de génotypage. La seconde direction est axée sur la plateforme, le projet peut servir de base pour l'implémentation des nouvelles fonctionnalités.

3.3.1 Modèle de données adam-ibs

Le modèle de données d'adam-ibs s'inspire du modèle de données de séquençage proposées par ADAM et propose une méthode pour couvrir les besoins en génotypage. Notamment, en plus des données génotypiques couvertes par ADAM, le modèle d'adam-ibs initie la couverture des données phénotypiques et introduit le concept des individus.

Le domaine de génotypage se concentre sur l'étude des individus associant leurs données génotypiques et phénotypiques. La méthodologie proposée par adam-ibs couvre cette approche et peut être complétée.

Lors d'une activité de recherche dans le cadre du projet Variantminer [24], aussi en partenariat avec le laboratoire du Dr Hamet, un scénario de réutilisation du modèle a été étudié. Les équipes de recherche, en génotypage, souhaitent effectuer des analyses sur des données issues des études cliniques. De même que dans l'approche proposée avec adam-ibs, les données de ces études sont axées sur les individus présentant des caractères génotypiques et phénotypiques. Les données phénotypiques sur les individus sont recueillies à l'occasion des visites programmées des patients chez les médecins réalisant l'étude.

La modélisation de ce cas d'application s'est inspirée du modèle proposé par adam-ibs mettant l'individu au centre des relations. La partie génotypique a été reprise du modèle proposé par Adam, et la partie phénotypique a été développée en introduisant le concept des visites permettant de recueillir les expressions phénotypiques des patients. La première version du modèle appliqué est proposée dans l'annexe VI.

3.3.2 Plateforme adam-ibs

Le paragraphe 2.3 détaille la structure, l'organisation, l'implémentation et les fonctionnalités de l'outil adam-ibs. Parmi les fonctionnalités, on distingue les fonctionnalités métier (représentés notamment par --make-bed et --genome) et les fonctionnalités de support telles que la gestion des logs, parcage de la ligne de commande, persistance, etc.

Ainsi, le projet présente une plateforme de développement assez complète pour pouvoir accueillir ces nouvelles fonctionnalités. Les fonctionnalités pouvant s'implémenter sont celles qui rentrent dans le cadre de l'un des trois processus présentés à la figure 6 : intégration des données, enrichissement des données et exploitation des données (c.-à-d. génération des rapports d'analyses).

Dans le cadre de projets de recherche futurs, il est donc plus simple et moins coûteux de réutiliser la plateforme technique d'adam-ibs pour implémenter les nouvelles fonctionnalités. À titre d'exemple, le projet de recherche Variantminer a pour l'objectif de mettre en place une interface d'utilisateur pour présenter les résultats d'analyses génotypiques. L'outil devra pouvoir exploiter les données sources présentées à l'annexe VI. L'un des prérequis incontournable de ce projet est d'avoir accès au fichier « Parquet » contenant les données cliniques sources. Pour ce faire, le projet aura besoin de mettre en place un ETL permettant d'extraire les données des sources actuelles (c.-à-d. les deux bases des données SQL qui regroupent l'ensemble des informations nécessaires aux analyses de l'étude clinique), former le modèle et les persister sur disque.

Dans le cadre des projets décisionnels, la mise en place d'un ETL peut prendre jusque 70% des coûts [25]. Dans le cas de Variantminer, l’effort nécessaire afin de réalise l’étape d’intégration des données ne doit pas non plus être sous-estimée. La qualité et l'intégrité des données à l'origine des requêtes d'analyse sont déterminantes pour l'acceptation du projet par les chercheurs. Il est donc important de constituer une base solide avant de procéder à l'exploitation et à la visualisation des données.

C'est dans ce contexte qu'il est possible d'exploiter la plateforme proposée par adam-ibs. L'intégration des données entre dans les trois processus supportés par le modèle (voir figure 6). Ainsi, au lieu développer totalement un nouveau module ETL spécifique à Variantminer, il est préférable de s'appuyer sur l'approche d'adam-ibs. Il suffirait seulement d'utiliser adam-ibs et de développer les fonctionnalités permettant d'intégrer les données cliniques dans le modèle parquet proposé.

3.4 Conclusion

La preuve de concept proposée par l'outil adam-ibs répond bien à la problématique initiale qui consistait à démontrer la faisabilité d'ajuster le modèle de données ADAM, de l'université Berkeley, pour inclure le génotypage. Au-delà d'une preuve de concept, le projet propose une méthodologie d'adaptation des modèles et des processus pour répondre à de nouveaux besoins. Notamment, l'outil adam-ibs peut être exploité davantage sur deux axes. Le premier axe consiste en une réadaptation du modèle proposé par adam-ibs centré sur le concept de l'individu. Le deuxième axe s'appuie sur la réutilisation de la plateforme proposée par adam-ibs pour la personnalisation et le développement des nouvelles fonctionnalités.

CONCLUSION

Le manque de normalisation du domaine de génotypage est un frein à la mise en place rapide d'outils génériques des traitements de grandes quantités de données. Le projet de recherche adam-ibs réalise une implémentation d'une partie des fonctionnalités de Plink, actuellement utilisé pour les études de type WGAS, proposant ainsi une méthodologie d'adaptation des modèles et des processus généralisés au domaine de génotypage. Cette démonstration valide l'hypothèse qu'il serait possible d'effectuer la migration d'outils comme Plink vers les technologies .mergentes du Big Data.

Adam-ibs s'appuie sur le modèle de données proposées par le projet ADAM et introduit le concept de l'individu faisant le lien entre les données génotypiques et phénotypiques. Par la suite, ce modèle a été validé à l'aide d'une étude de cas pour répondre aux besoins de l'équipe des chercheurs du Dr Hamet au CRCHUM. En introduisant le concept des visites et en développant les paramètres phénotypiques, le modèle a pu facilement s'adapter et couvrir entièrement un contexte d'étude clinique existante.

En tant que plateforme, adam-ibs définit et supporte les processus d'intégration, d'enrichissement et d'exploitation des données. Cette proposition peut être utilisée pour le développement de nouvelles fonctionnalités. La réutilisation d'adam-ibs permettrait de réduire les temps de développement des différents projets en bio informatique du laboratoire.

ANNEXE I

Intégration de Parquet avec Avro

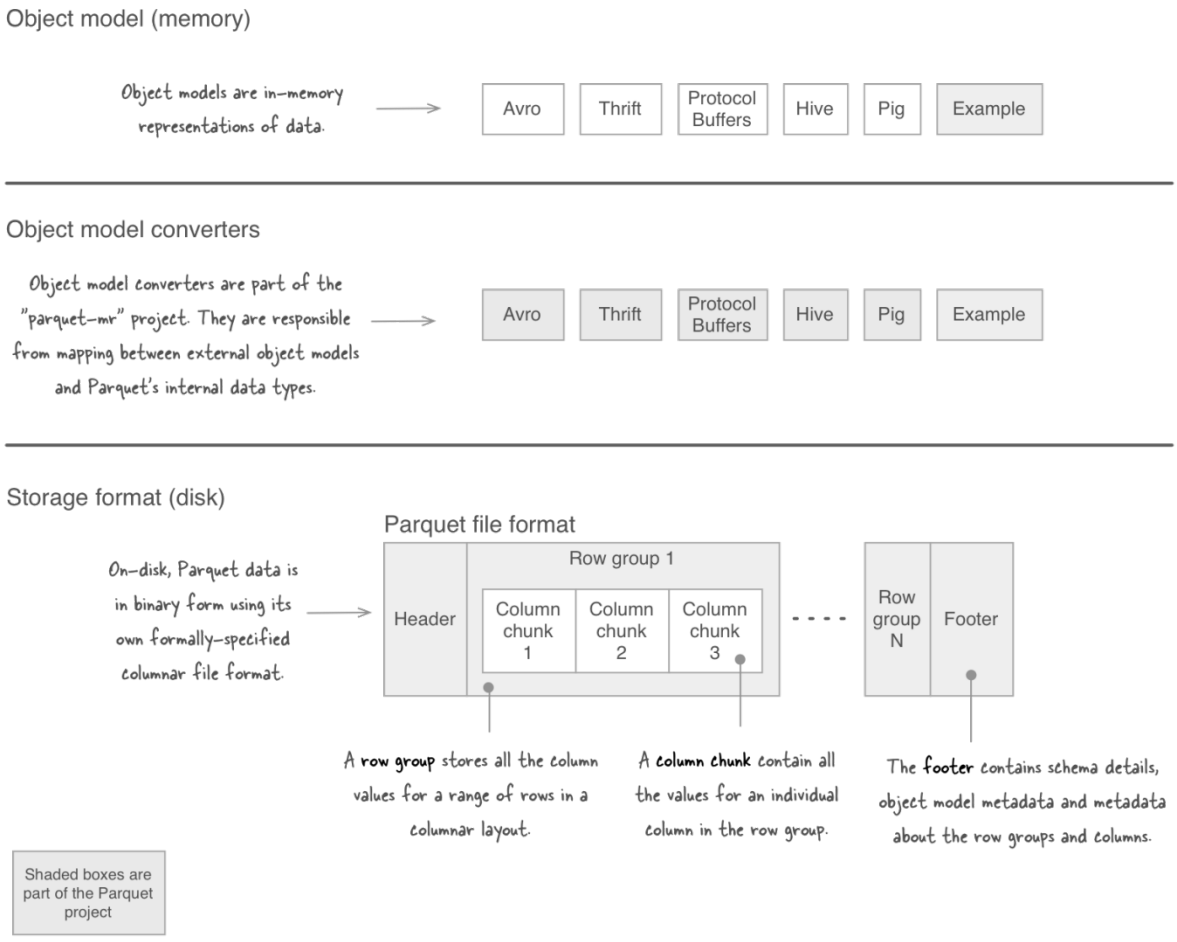

Figure 9 : Intégration de parquet avec Avro [16]

ANNEXE II

Modèle de donnés ADAM bdg-formats

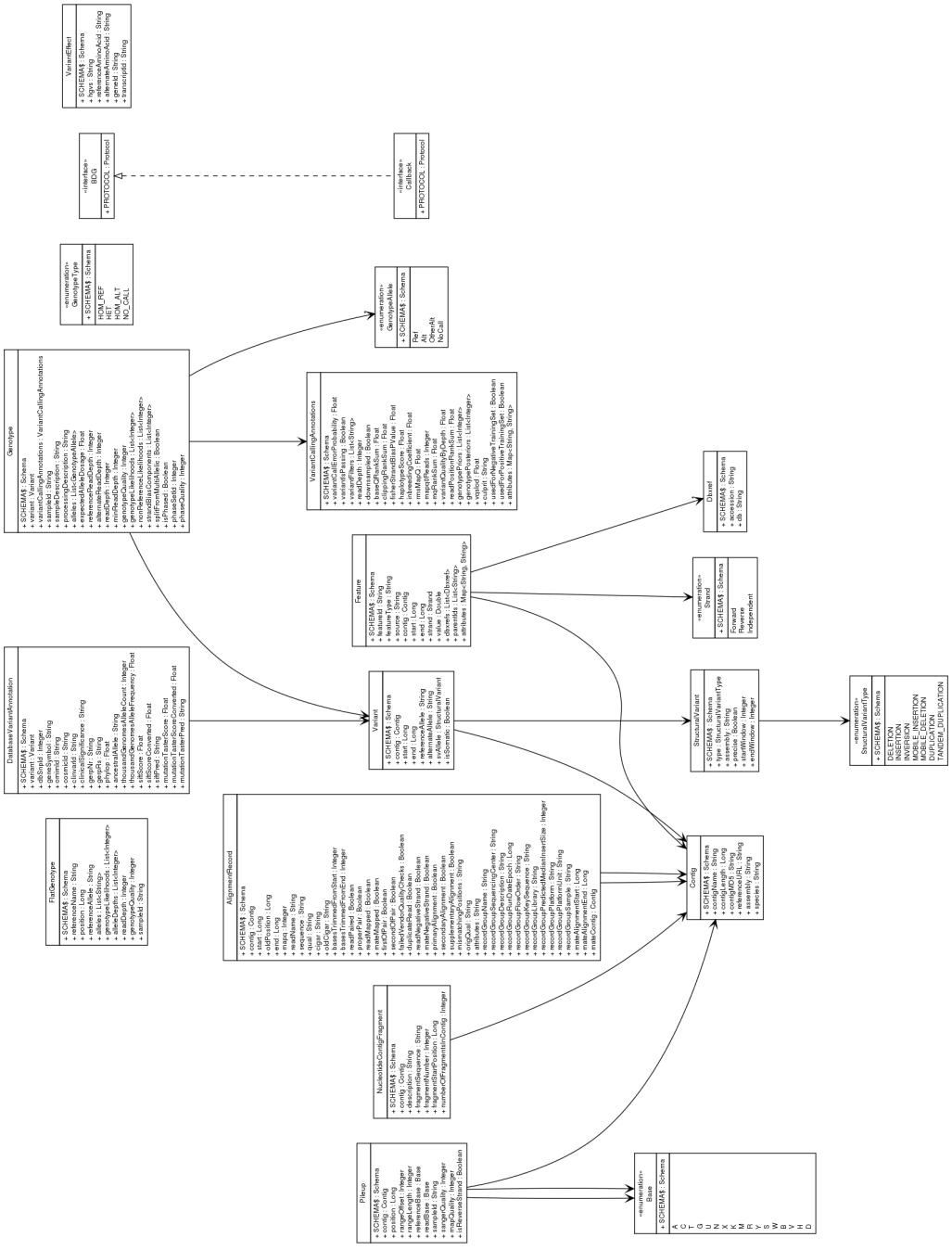

Figure 10 : Modèle de données ADAM bgd-formats [17]

ANNEXE III

Formats de données Plink

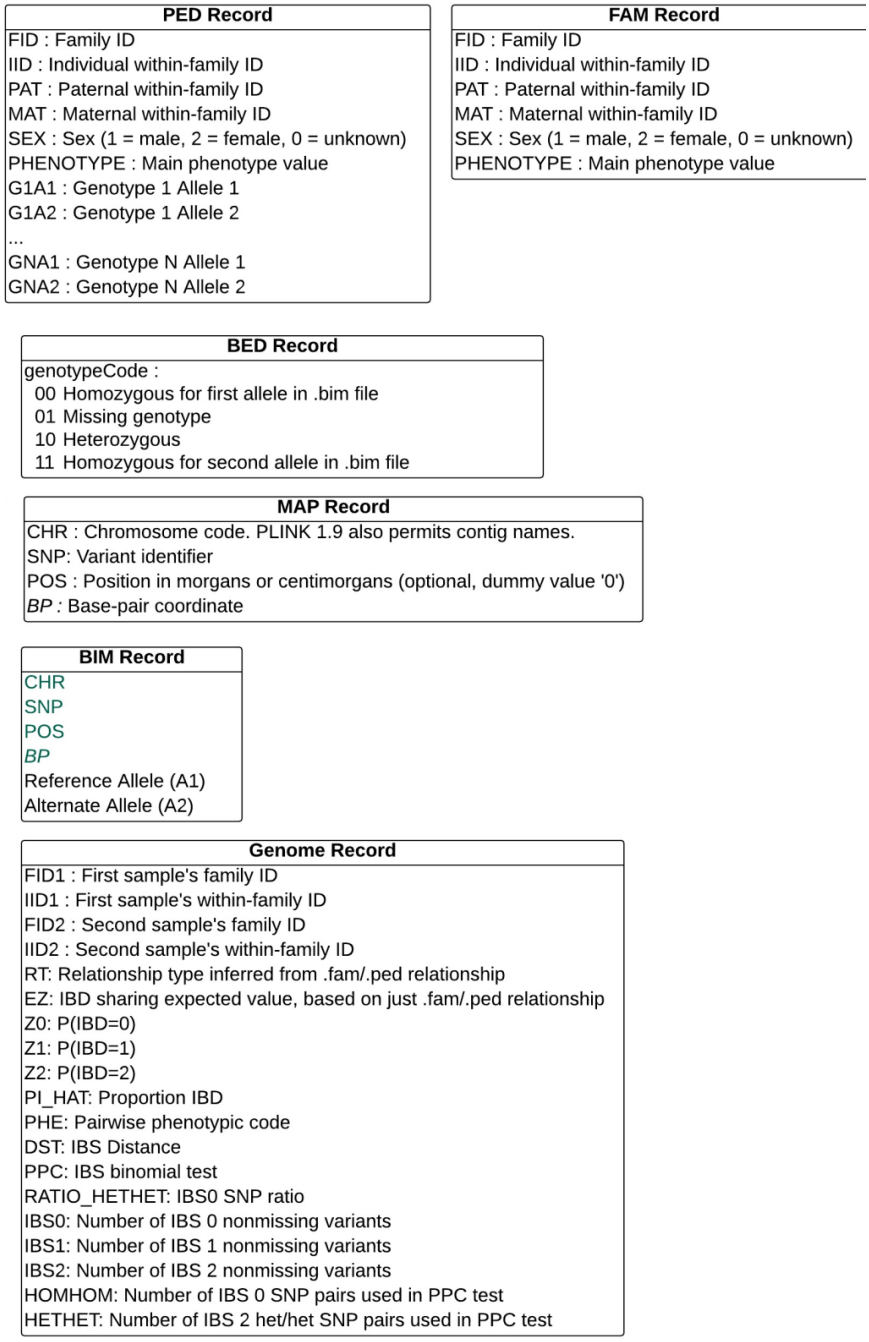

Figure 11 : Formats de données de Plink

ANNEXE IV

Intégration des besoins de Plink dans le modèle des données de ADAM

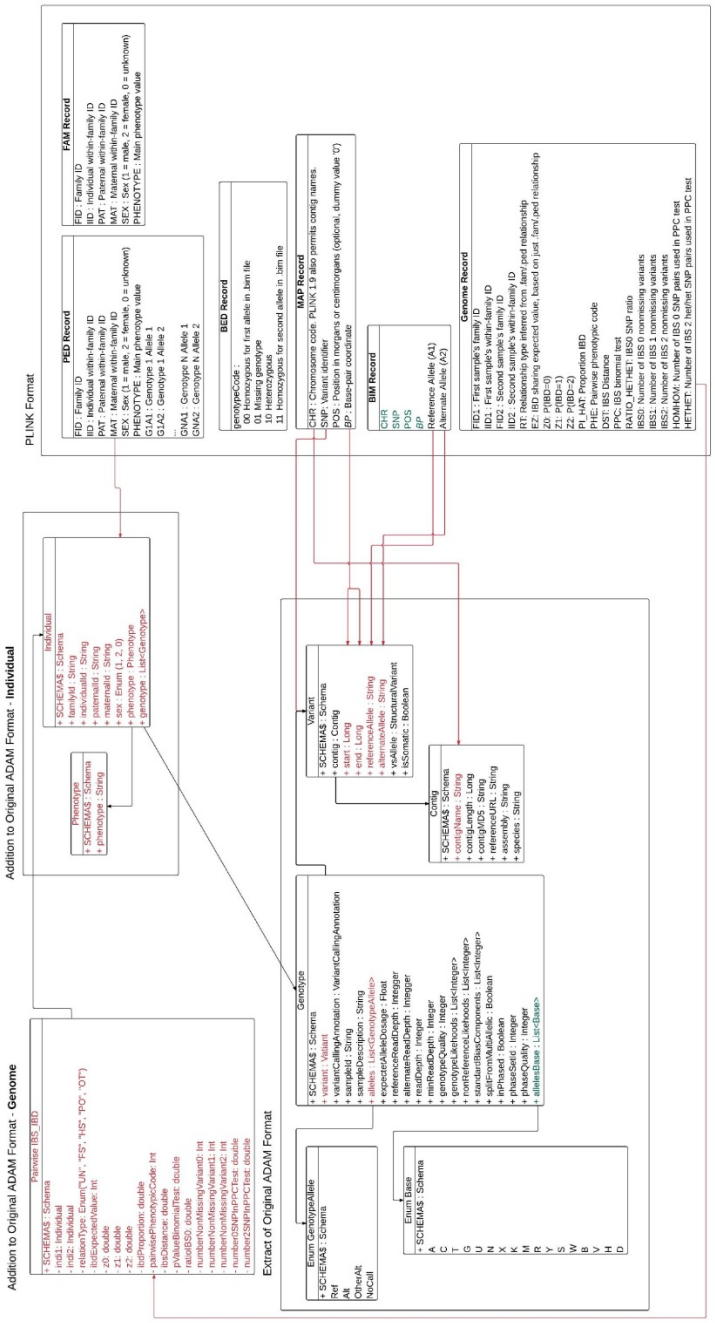

Figure 12 : Modèle de données intégrant les besoins de Plink

ANNEXE V

Données d'exécution d'adam-ibs

Données sources test.ped et test.map :

test.ped :

```
1 1 0 0 1 1 A A G T
2 1 0 0 1 1 A C T G
3 1 0 0 1 1 C C G G
4 1 0 0 1 2 A C T T
5 1 0 0 1 2 C C G T
6 1 0 0 1 2 C C T T
```

test.map :

```
1 snp1 0 1
1 snp2 0 2
```

Log exécution --make-bed (fichier process.MakeBed.log) :

```
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.InternalParquetRecordWriter: Flushing mem columnStore to file. allocated memory: 101,934,218
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 63B for [familyId] BINARY: 6 values, 36B raw, 36B comp, 1 pages, encodings: [RLE, BIT_PACKED, PLAIN]
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 35B for [individualId] BINARY: 6 values, 8B raw, 8B comp, 1 pages, encodings: [RLE, BIT_PACKED, PLAIN_DICTIONARY], dic { 1 entries, 5B raw, 1B comp}
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 35B for [paternalId] BINARY: 6 values, 8B raw, 8B comp, 1 pages, encodings: [RLE, BIT_PACKED, PLAIN_DICTIONARY], dic { 1 entries, 5B raw, 1B comp}
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 35B for [maternalId] BINARY: 6 values, 8B raw, 8B comp, 1 pages, encodings: [RLE, BIT_PACKED, PLAIN_DICTIONARY], dic { 1 entries, 5B raw, 1B comp}
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 41B for [sex] BINARY: 6 values, 8B raw, 8B comp, 1 pages, encodings: [RLE, BIT_PACKED, PLAIN_DICTIONARY], dic { 1 entries, 8B raw, 1B comp}
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 37B for [phenotype, phenotype] BINARY: 6 values, 10B raw, 10B comp, 1 pages, encodings: [RLE, BIT_PACKED, PLAIN_DICTIONARY], dic { 2 entries, 10B raw, 2B comp}
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 42B for [genotype, array, variant, contig, contigName] BINARY: 12 values, 15B raw, 15B comp, 1 pages, encodings: [RLE, PLAIN_DICTIONARY], dic { 1 entries, 5B raw, 1B comp}
```

```
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 30B for [genotype, array, variant, contig, contigLength] INT64: 12 values, 13B raw, 13B comp, 1 pages, encodings: [RLE, PLAIN]
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 30B for [genotype, array, variant, contig, contigMD5] BINARY: 12 values, 13B raw, 13B comp, 1 pages, encodings: [RLE, PLAIN]
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 30B for [genotype, array, variant, contig, referenceURL] BINARY: 12 values, 13B raw, 13B comp, 1 pages, encodings: [RLE, PLAIN]
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 30B for [genotype, array, variant, contig, assembly] BINARY: 12 values, 13B raw, 13B comp, 1 pages, encodings: [RLE, PLAIN]
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 30B for [genotype, array, variant, contig, species] BINARY: 12 values, 13B raw, 13B comp, 1 pages, encodings: [RLE, PLAIN]
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 58B for [genotype, array, variant, start] INT64: 12 values, 17B raw, 17B comp, 1 pages, encodings: [RLE, PLAIN_DICTIONARY], dic { 2 entries, 16B raw, 2B comp}
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 58B for [genotype, array, variant, end] INT64: 12 values, 17B raw, 17B comp, 1 pages, encodings: [RLE, PLAIN_DICTIONARY], dic { 2 entries, 16B raw, 2B comp}
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 44B for [genotype, array, variant, referenceAllele] BINARY: 12 values, 17B raw, 17B comp, 1 pages, encodings: [RLE, PLAIN_DICTIONARY], dic { 2 entries, 10B raw, 2B comp}
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 44B for [genotype, array, variant, alternateAllele] BINARY: 12 values, 17B raw, 17B comp, 1 pages, encodings: [RLE, PLAIN_DICTIONARY], dic { 2 entries, 10B raw, 2B comp}
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 30B for [genotype, array, variant, svAllele, type] BINARY: 12 values, 13B raw, 13B comp, 1 pages, encodings: [RLE, PLAIN]
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 30B for [genotype, array, variant, svAllele, assembly] BINARY: 12 values, 13B raw, 13B comp, 1 pages, encodings: [RLE, PLAIN]
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 30B for [genotype, array, variant, svAllele, precise] BOOLEAN: 12 values, 13B raw, 13B comp, 1 pages, encodings: [RLE, PLAIN]
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 30B for [genotype, array, variant, svAllele, startWindow] INT32: 12 values, 13B raw, 13B comp, 1 pages, encodings: [RLE, PLAIN]
```

```
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 30B for [genotype, array, variant, svAllele, endWindow] INT32: 12 values, 13B raw, 13B comp, 1 pages, encodings: [RLE, PLAIN]
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 30B for [genotype, array, variant, isSomatic] BOOLEAN: 12 values, 13B raw, 13B comp, 1 pages, encodings: [RLE, PLAIN]
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 30B for [genotype, array, variantCallingAnnotations, variantCallErrorProbability] FLOAT: 12 values, 13B raw, 13B comp, 1 pages, encodings: [RLE, PLAIN]
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 30B for [genotype, array, variantCallingAnnotations, variantIsPassing] BOOLEAN: 12 values, 13B raw, 13B comp, 1 pages, encodings: [RLE, PLAIN]
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [genotype, array, variantCallingAnnotations, variantFilters, array] BINARY: 12 values, 15B raw, 15B comp, 1 pages, encodings: [RLE, PLAIN]
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 30B for [genotype, array, variantCallingAnnotations, readDepth] INT32: 12 values, 13B raw, 13B comp, 1 pages, encodings: [RLE, PLAIN]
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 30B for [genotype, array, variantCallingAnnotations, downsampled] BOOLEAN: 12 values, 13B raw, 13B comp, 1 pages, encodings: [RLE, PLAIN]
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 30B for [genotype, array, variantCallingAnnotations, baseQRankSum] FLOAT: 12 values, 13B raw, 13B comp, 1 pages, encodings: [RLE, PLAIN]
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 30B for [genotype, array, variantCallingAnnotations, clippingRankSum] FLOAT: 12 values, 13B raw, 13B comp, 1 pages, encodings: [RLE, PLAIN]
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 30B for [genotype, array, variantCallingAnnotations, fisherStrandBiasPValue] FLOAT: 12 values, 13B raw, 13B comp, 1 pages, encodings: [RLE, PLAIN]
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 30B for [genotype, array, variantCallingAnnotations, haplotypeScore] FLOAT: 12 values, 13B raw, 13B comp, 1 pages, encodings: [RLE, PLAIN]
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 30B for [genotype, array, variantCallingAnnotations, inbreedingCoefficient] FLOAT: 12 values, 13B raw, 13B comp, 1 pages, encodings: [RLE, PLAIN]
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 30B for [genotype, array, variantCallingAnnotations, rmsMapQ] FLOAT: 12 values, 13B raw, 13B comp, 1 pages, encodings: [RLE, PLAIN]
```

```
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 30B for [genotype, array, variantCallingAnnotations, mapq0Reads] INT32: 12 values, 13B raw, 13B comp, 1 pages, encodings: [RLE, PLAIN]
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 30B for [genotype, array, variantCallingAnnotations, mqRankSum] FLOAT: 12 values, 13B raw, 13B comp, 1 pages, encodings: [RLE, PLAIN]
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 30B for [genotype, array, variantCallingAnnotations, variantQualityByDepth] FLOAT: 12 values, 13B raw, 13B comp, 1 pages, encodings: [RLE, PLAIN]
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 30B for [genotype, array, variantCallingAnnotations, readPositionRankSum] FLOAT: 12 values, 13B raw, 13B comp, 1 pages, encodings: [RLE, PLAIN]
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [genotype, array, variantCallingAnnotations, genotypePriors, array] INT32: 12 values, 15B raw, 15B comp, 1 pages, encodings: [RLE, PLAIN]
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [genotype, array, variantCallingAnnotations, genotypePosteriors, array] INT32: 12 values, 15B raw, 15B comp, 1 pages, encodings: [RLE, PLAIN]
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 30B for [genotype, array, variantCallingAnnotations, vqslod] FLOAT: 12 values, 13B raw, 13B comp, 1 pages, encodings: [RLE, PLAIN]
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 30B for [genotype, array, variantCallingAnnotations, culprit] BINARY: 12 values, 13B raw, 13B comp, 1 pages, encodings: [RLE, PLAIN]
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 30B for [genotype, array, variantCallingAnnotations, usedForNegativeTrainingSet] BOOLEAN: 12 values, 13B raw, 13B comp, 1 pages, encodings: [RLE, PLAIN]
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 30B for [genotype, array, variantCallingAnnotations, usedForPositiveTrainingSet] BOOLEAN: 12 values, 13B raw, 13B comp, 1 pages, encodings: [RLE, PLAIN]
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [genotype, array, variantCallingAnnotations, attributes, map, key] BINARY: 12 values, 15B raw, 15B comp, 1 pages, encodings: [RLE, PLAIN]
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [genotype, array, variantCallingAnnotations, attributes, map, value] BINARY: 12 values, 15B raw, 15B comp, 1 pages, encodings: [RLE, PLAIN]
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 30B for [genotype, array, sampleId] BINARY: 12 values, 13B raw, 13B comp, 1 pages, encodings: [RLE, PLAIN]
```

```
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 30B for [genotype, array, sampleDescription] BINARY: 12 values, 13B raw, 13B comp, 1 pages, encodings: [RLE, PLAIN]
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 30B for [genotype, array, processingDescription] BINARY: 12 values, 13B raw, 13B comp, 1 pages, encodings: [RLE, PLAIN]
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 53B for [genotype, array, alleles, array] BINARY: 24 values, 22B raw, 22B comp, 1 pages, encodings: [RLE, PLAIN_DICTIONARY], dic { 2 entries, 14B raw, 2B comp}
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 30B for [genotype, array, expectedAlleleDosage] FLOAT: 12 values, 13B raw, 13B comp, 1 pages, encodings: [RLE, PLAIN]
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 30B for [genotype, array, referenceReadDepth] INT32: 12 values, 13B raw, 13B comp, 1 pages, encodings: [RLE, PLAIN]
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 30B for [genotype, array, alternateReadDepth] INT32: 12 values, 13B raw, 13B comp, 1 pages, encodings: [RLE, PLAIN]
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 30B for [genotype, array, readDepth] INT32: 12 values, 13B raw, 13B comp, 1 pages, encodings: [RLE, PLAIN]
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 30B for [genotype, array, minReadDepth] INT32: 12 values, 13B raw, 13B comp, 1 pages, encodings: [RLE, PLAIN]
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 30B for [genotype, array, genotypeQuality] INT32: 12 values, 13B raw, 13B comp, 1 pages, encodings: [RLE, PLAIN]
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [genotype, array, genotypeLikelihoods, array] INT32: 12 values, 15B raw, 15B comp, 1 pages, encodings: [RLE, PLAIN]
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [genotype, array, nonReferenceLikelihoods, array] INT32: 12 values, 15B raw, 15B comp, 1 pages, encodings: [RLE, PLAIN]
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [genotype, array, strandBiasComponents, array] INT32: 12 values, 15B raw, 15B comp, 1 pages, encodings: [RLE, PLAIN]
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 30B for [genotype, array, splitFromMultiAllelic] BOOLEAN: 12 values, 13B raw, 13B comp, 1 pages, encodings: [RLE, PLAIN]
```

```
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 30B for [genotype, array, isPhased] BOOLEAN: 12 values, 13B raw, 13B comp, 1 pages, encodings: [RLE, PLAIN]
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 30B for [genotype, array, phaseSetId] INT32: 12 values, 13B raw, 13B comp, 1 pages, encodings: [RLE, PLAIN]
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 30B for [genotype, array, phaseQuality] INT32: 12 values, 13B raw, 13B comp, 1 pages, encodings: [RLE, PLAIN]
4-Nov-2015 5:19:28 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 52B for [genotype, array, allelesBase, array] BINARY: 24 values, 25B raw, 25B comp, 1 pages, encodings: [RLE, PLAIN_DICTIONARY], dic { 4 entries, 20B raw, 4B comp}
```

Log exécution --show-parquet sur le fichier obtenu par --make-bed (fichier show.testMakeBed) :

```
root
 |-- familyId: string (nullable = true)
 |-- individualId: string (nullable = true)
 |-- paternalId: string (nullable = true)
 |-- maternalId: string (nullable = true)
 |-- sex: string (nullable = true)
 |-- phenotype: struct (nullable = true)
 |    |-- phenotype: string (nullable = true)
 |-- genotype: array (nullable = true)
 |    |-- element: struct (containsNull = false)
 |    |    |-- variant: struct (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)
 |    |    |    |-- 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)
|    |    |    |-- variantCallErrorProbability: float (nullable = true)
|    |    |    |-- variantIsPassing: boolean (nullable = true)
|    |    |    |-- variantFilters: array (nullable = false)
|    |    |    |    |-- element: string (containsNull = false)
|    |    |    |-- readDepth: integer (nullable = true)
|    |    |    |-- downsampled: boolean (nullable = true)
|    |    |    |-- baseQRankSum: float (nullable = true)
|    |    |    |-- clippingRankSum: float (nullable = true)
|    |    |    |-- fisherStrandBiasPValue: float (nullable = true)
|    |    |    |-- haplotypeScore: float (nullable = true)
|    |    |    |-- inbreedingCoefficient: float (nullable = true)
|    |    |    |-- rmsMapQ: float (nullable = true)
|    |    |    |-- mapq0Reads: integer (nullable = true)
|    |    |    |-- mqRankSum: float (nullable = true)
|    |    |    |-- variantQualityByDepth: float (nullable = true)
|    |    |    |-- readPositionRankSum: float (nullable = true)
|    |    |    |-- genotypePriors: array (nullable = false)
|    |    |    |    |-- element: integer (containsNull = false)
|    |    |    |-- genotypePosteriors: array (nullable = false)
|    |    |    |    |-- element: integer (containsNull = false)
|    |    |    |-- vqslod: float (nullable = true)
|    |    |    |-- culprit: string (nullable = true)
|    |    |    |-- usedForNegativeTrainingSet: boolean (nullable = true)
|    |    |    |-- usedForPositiveTrainingSet: boolean (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 = true)
|    |    |    |-- 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 = true)
 |    |    |    |-- element: integer (containsNull = false)
 |    |    |-- nonReferenceLikelihoods: array (nullable = true)
 |    |    |    |-- element: integer (containsNull = false)
 |    |    |-- strandBiasComponents: array (nullable = true)
 |    |    |    |-- element: integer (containsNull = false)
 |    |    |-- splitFromMultiAllelic: boolean (nullable = true)
 |    |    |-- isPhased: boolean (nullable = true)
 |    |    |-- phaseSetId: integer (nullable = true)
 |    |    |-- phaseQuality: integer (nullable = true)
 |    |    |-- allelesBase: array (nullable = true)
 |    |    |    |-- element: string (containsNull = false)

+--------+------------+----------+----------+----+---------+--------------------+
|familyId|individualId|paternalId|maternalId| sex|phenotype|            genotype|
+--------+------------+----------+----------+----+---------+--------------------+
|       1|           1|         0|         0|Male|      [1]|ArrayBuffer([[[1,...|
|       2|           1|         0|         0|Male|      [1]|ArrayBuffer([[[1,...|
|       3|           1|         0|         0|Male|      [1]|ArrayBuffer([[[1,...|
|       4|           1|         0|         0|Male|      [2]|ArrayBuffer([[[1,...|
|       5|           1|         0|         0|Male|      [2]|ArrayBuffer([[[1,...|
|       6|           1|         0|         0|Male|      [2]|ArrayBuffer([[[1,...|
+--------+------------+----------+----------+----+---------+--------------------+

4-Nov-2015 5:28:40 PM INFO: parquet.hadoop.ParquetFileReader: Initiating action
with parallelism: 5
4-Nov-2015 5:28:42 PM WARNING: parquet.hadoop.ParquetRecordReader: Can not
initialize counter due to context is not a instance of TaskInputOutputContext, but
is org.apache.hadoop.mapreduce.task.TaskAttemptContextImpl
4-Nov-2015 5:28:42 PM INFO: parquet.hadoop.InternalParquetRecordReader:
RecordReader initialized will read a total of 6 records.
4-Nov-2015 5:28:42 PM INFO: parquet.hadoop.InternalParquetRecordReader: at row 0.
reading next block
4-Nov-2015 5:28:42 PM INFO: parquet.hadoop.InternalParquetRecordReader: block read
in memory in 44 ms. row count = 6
```

Log exécution --genome (fichier process.Genome.log) :

```
4-Nov-2015 5:24:05 PM INFO: parquet.hadoop.ParquetFileReader: Initiating action with parallelism: 5
4-Nov-2015 5:24:05 PM INFO: parquet.hadoop.ParquetFileReader: reading another 1 footers
4-Nov-2015 5:24:05 PM INFO: parquet.hadoop.ParquetFileReader: Initiating action with parallelism: 5
4-Nov-2015 5:24:05 PM INFO: parquet.hadoop.InternalParquetRecordReader: RecordReader initialized will read a total of 6 records.
4-Nov-2015 5:24:05 PM INFO: parquet.hadoop.InternalParquetRecordReader: at row 0. reading next block
4-Nov-2015 5:24:05 PM INFO: parquet.hadoop.InternalParquetRecordReader: block read in memory in 22 ms. row count = 6
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.InternalParquetRecordWriter: Flushing mem columnStore to file. allocated memory: 100,239,859
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 41B for [individual1, familyId] BINARY: 15 values, 14B raw, 14B comp, 1 pages, encodings: [BIT_PACKED, RLE, PLAIN_DICTIONARY], dic { 5 entries, 25B raw, 5B comp}
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 35B for [individual1, individualId] BINARY: 15 values, 8B raw, 8B comp, 1 pages, encodings: [BIT_PACKED, RLE, PLAIN_DICTIONARY], dic { 1 entries, 5B raw, 1B comp}
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 35B for [individual1, paternalId] BINARY: 15 values, 8B raw, 8B comp, 1 pages, encodings: [BIT_PACKED, RLE, PLAIN_DICTIONARY], dic { 1 entries, 5B raw, 1B comp}
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 35B for [individual1, maternalId] BINARY: 15 values, 8B raw, 8B comp, 1 pages, encodings: [BIT_PACKED, RLE, PLAIN_DICTIONARY], dic { 1 entries, 5B raw, 1B comp}
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 41B for [individual1, sex] BINARY: 15 values, 8B raw, 8B comp, 1 pages, encodings: [BIT_PACKED, RLE, PLAIN_DICTIONARY], dic { 1 entries, 8B raw, 1B comp}
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 38B for [individual1, phenotype, phenotype] BINARY: 15 values, 11B raw, 11B comp, 1 pages, encodings: [BIT_PACKED, RLE, PLAIN_DICTIONARY], dic { 2 entries, 10B raw, 2B comp}
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 44B for [individual1, genotype, array, variant, contig, contigName] BINARY: 30 values, 17B raw, 17B comp, 1 pages, encodings: [RLE, PLAIN_DICTIONARY], dic { 1 entries, 5B raw, 1B comp}
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual1, genotype, array, variant, contig, contigLength] INT64: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
```

```
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual1, genotype, array, variant, contig, contigMD5] BINARY: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual1, genotype, array, variant, contig, referenceURL] BINARY: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual1, genotype, array, variant, contig, assembly] BINARY: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual1, genotype, array, variant, contig, species] BINARY: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 62B for [individual1, genotype, array, variant, start] INT64: 30 values, 21B raw, 21B comp, 1 pages, encodings: [RLE, PLAIN_DICTIONARY], dic { 2 entries, 16B raw, 2B comp}
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 62B for [individual1, genotype, array, variant, end] INT64: 30 values, 21B raw, 21B comp, 1 pages, encodings: [RLE, PLAIN_DICTIONARY], dic { 2 entries, 16B raw, 2B comp}
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 48B for [individual1, genotype, array, variant, referenceAllele] BINARY: 30 values, 21B raw, 21B comp, 1 pages, encodings: [RLE, PLAIN_DICTIONARY], dic { 2 entries, 10B raw, 2B comp}
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 48B for [individual1, genotype, array, variant, alternateAllele] BINARY: 30 values, 21B raw, 21B comp, 1 pages, encodings: [RLE, PLAIN_DICTIONARY], dic { 2 entries, 10B raw, 2B comp}
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual1, genotype, array, variant, svAllele, type] BINARY: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual1, genotype, array, variant, svAllele, assembly] BINARY: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual1, genotype, array, variant, svAllele, precise] BOOLEAN: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual1, genotype, array, variant, svAllele, startWindow] INT32: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
```

```
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual1, genotype, array, variant, svAllele, endWindow] INT32: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual1, genotype, array, variant, isSomatic] BOOLEAN: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual1, genotype, array, variantCallingAnnotations, variantCallErrorProbability] FLOAT: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual1, genotype, array, variantCallingAnnotations, variantIsPassing] BOOLEAN: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 36B for [individual1, genotype, array, variantCallingAnnotations, variantFilters, array] BINARY: 30 values, 19B raw, 19B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual1, genotype, array, variantCallingAnnotations, readDepth] INT32: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual1, genotype, array, variantCallingAnnotations, downsampled] BOOLEAN: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual1, genotype, array, variantCallingAnnotations, baseQRankSum] FLOAT: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual1, genotype, array, variantCallingAnnotations, clippingRankSum] FLOAT: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual1, genotype, array, variantCallingAnnotations, fisherStrandBiasPValue] FLOAT: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual1, genotype, array, variantCallingAnnotations, haplotypeScore] FLOAT: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual1, genotype, array, variantCallingAnnotations, inbreedingCoefficient] FLOAT: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
```

```
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual1, genotype, array, variantCallingAnnotations, rmsMapQ] FLOAT: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual1, genotype, array, variantCallingAnnotations, mapq0Reads] INT32: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual1, genotype, array, variantCallingAnnotations, mqRankSum] FLOAT: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual1, genotype, array, variantCallingAnnotations, variantQualityByDepth] FLOAT: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual1, genotype, array, variantCallingAnnotations, readPositionRankSum] FLOAT: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 36B for [individual1, genotype, array, variantCallingAnnotations, genotypePriors, array] INT32: 30 values, 19B raw, 19B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 36B for [individual1, genotype, array, variantCallingAnnotations, genotypePosteriors, array] INT32: 30 values, 19B raw, 19B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual1, genotype, array, variantCallingAnnotations, vqslod] FLOAT: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual1, genotype, array, variantCallingAnnotations, culprit] BINARY: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual1, genotype, array, variantCallingAnnotations, usedForNegativeTrainingSet] BOOLEAN: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual1, genotype, array, variantCallingAnnotations, usedForPositiveTrainingSet] BOOLEAN: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 36B for [individual1, genotype, array, variantCallingAnnotations, attributes, map, key] BINARY: 30 values, 19B raw, 19B comp, 1 pages, encodings: [PLAIN, RLE]
```

```
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 36B for [individual1, genotype, array, variantCallingAnnotations, attributes, map, value] BINARY: 30 values, 19B raw, 19B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual1, genotype, array, sampleId] BINARY: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual1, genotype, array, sampleDescription] BINARY: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual1, genotype, array, processingDescription] BINARY: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 68B for [individual1, genotype, array, alleles, array] BINARY: 60 values, 37B raw, 37B comp, 1 pages, encodings: [RLE, PLAIN_DICTIONARY], dic { 2 entries, 14B raw, 2B comp}
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual1, genotype, array, expectedAlleleDosage] FLOAT: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual1, genotype, array, referenceReadDepth] INT32: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual1, genotype, array, alternateReadDepth] INT32: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual1, genotype, array, readDepth] INT32: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual1, genotype, array, minReadDepth] INT32: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual1, genotype, array, genotypeQuality] INT32: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 36B for [individual1, genotype, array, genotypeLikelihoods, array] INT32: 30 values, 19B raw, 19B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 36B for [individual1, genotype, array, nonReferenceLikelihoods, array] INT32: 30 values, 19B raw, 19B comp, 1 pages, encodings: [PLAIN, RLE]
```

```
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 36B for [individual1, genotype, array, strandBiasComponents, array] INT32: 30 values, 19B raw, 19B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual1, genotype, array, splitFromMultiAllelic] BOOLEAN: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual1, genotype, array, isPhased] BOOLEAN: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual1, genotype, array, phaseSetId] INT32: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual1, genotype, array, phaseQuality] INT32: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 72B for [individual1, genotype, array, allelesBase, array] BINARY: 60 values, 45B raw, 45B comp, 1 pages, encodings: [RLE, PLAIN_DICTIONARY], dic { 4 entries, 20B raw, 4B comp}
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 41B for [individual2, familyId] BINARY: 15 values, 14B raw, 14B comp, 1 pages, encodings: [BIT_PACKED, RLE, PLAIN_DICTIONARY], dic { 5 entries, 25B raw, 5B comp}
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 35B for [individual2, individualId] BINARY: 15 values, 8B raw, 8B comp, 1 pages, encodings: [BIT_PACKED, RLE, PLAIN_DICTIONARY], dic { 1 entries, 5B raw, 1B comp}
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 35B for [individual2, paternalId] BINARY: 15 values, 8B raw, 8B comp, 1 pages, encodings: [BIT_PACKED, RLE, PLAIN_DICTIONARY], dic { 1 entries, 5B raw, 1B comp}
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 35B for [individual2, maternalId] BINARY: 15 values, 8B raw, 8B comp, 1 pages, encodings: [BIT_PACKED, RLE, PLAIN_DICTIONARY], dic { 1 entries, 5B raw, 1B comp}
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 41B for [individual2, sex] BINARY: 15 values, 8B raw, 8B comp, 1 pages, encodings: [BIT_PACKED, RLE, PLAIN_DICTIONARY], dic { 1 entries, 8B raw, 1B comp}
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 37B for [individual2, phenotype, phenotype] BINARY: 15 values, 10B raw, 10B comp, 1 pages, encodings: [BIT_PACKED, RLE, PLAIN_DICTIONARY], dic { 2 entries, 10B raw, 2B comp}
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 44B for [individual2, genotype, array, variant, contig, contigName] BINARY: 30 values,
```

```
17B raw, 17B comp, 1 pages, encodings: [RLE, PLAIN_DICTIONARY], dic { 1 entries, 5B raw, 1B comp}
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual2, genotype, array, variant, contig, contigLength] INT64: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual2, genotype, array, variant, contig, contigMD5] BINARY: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual2, genotype, array, variant, contig, referenceURL] BINARY: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual2, genotype, array, variant, contig, assembly] BINARY: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual2, genotype, array, variant, contig, species] BINARY: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 62B for [individual2, genotype, array, variant, start] INT64: 30 values, 21B raw, 21B comp, 1 pages, encodings: [RLE, PLAIN_DICTIONARY], dic { 2 entries, 16B raw, 2B comp}
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 62B for [individual2, genotype, array, variant, end] INT64: 30 values, 21B raw, 21B comp, 1 pages, encodings: [RLE, PLAIN_DICTIONARY], dic { 2 entries, 16B raw, 2B comp}
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 48B for [individual2, genotype, array, variant, referenceAllele] BINARY: 30 values, 21B raw, 21B comp, 1 pages, encodings: [RLE, PLAIN_DICTIONARY], dic { 2 entries, 10B raw, 2B comp}
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 48B for [individual2, genotype, array, variant, alternateAllele] BINARY: 30 values, 21B raw, 21B comp, 1 pages, encodings: [RLE, PLAIN_DICTIONARY], dic { 2 entries, 10B raw, 2B comp}
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual2, genotype, array, variant, svAllele, type] BINARY: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual2, genotype, array, variant, svAllele, assembly] BINARY: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
```

```
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual2, genotype, array, variant, svAllele, precise] BOOLEAN: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual2, genotype, array, variant, svAllele, startWindow] INT32: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual2, genotype, array, variant, svAllele, endWindow] INT32: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual2, genotype, array, variant, isSomatic] BOOLEAN: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual2, genotype, array, variantCallingAnnotations, variantCallErrorProbability] FLOAT: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual2, genotype, array, variantCallingAnnotations, variantIsPassing] BOOLEAN: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 36B for [individual2, genotype, array, variantCallingAnnotations, variantFilters, array] BINARY: 30 values, 19B raw, 19B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual2, genotype, array, variantCallingAnnotations, readDepth] INT32: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual2, genotype, array, variantCallingAnnotations, downsampled] BOOLEAN: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual2, genotype, array, variantCallingAnnotations, baseQRankSum] FLOAT: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual2, genotype, array, variantCallingAnnotations, clippingRankSum] FLOAT: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual2, genotype, array, variantCallingAnnotations, fisherStrandBiasPValue] FLOAT: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual2, genotype, array, variantCallingAnnotations, haplotypeScore] FLOAT: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
```

```
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual2, genotype, array, variantCallingAnnotations, inbreedingCoefficient] FLOAT: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual2, genotype, array, variantCallingAnnotations, rmsMapQ] FLOAT: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual2, genotype, array, variantCallingAnnotations, mapq0Reads] INT32: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual2, genotype, array, variantCallingAnnotations, mqRankSum] FLOAT: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual2, genotype, array, variantCallingAnnotations, variantQualityByDepth] FLOAT: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual2, genotype, array, variantCallingAnnotations, readPositionRankSum] FLOAT: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 36B for [individual2, genotype, array, variantCallingAnnotations, genotypePriors, array] INT32: 30 values, 19B raw, 19B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 36B for [individual2, genotype, array, variantCallingAnnotations, genotypePosteriors, array] INT32: 30 values, 19B raw, 19B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual2, genotype, array, variantCallingAnnotations, vqslod] FLOAT: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual2, genotype, array, variantCallingAnnotations, culprit] BINARY: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual2, genotype, array, variantCallingAnnotations, usedForNegativeTrainingSet] BOOLEAN: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual2, genotype, array, variantCallingAnnotations, usedForPositiveTrainingSet] BOOLEAN: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
```

```
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 36B for [individual2, genotype, array, variantCallingAnnotations, attributes, map, key] BINARY: 30 values, 19B raw, 19B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 36B for [individual2, genotype, array, variantCallingAnnotations, attributes, map, value] BINARY: 30 values, 19B raw, 19B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual2, genotype, array, sampleId] BINARY: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual2, genotype, array, sampleDescription] BINARY: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual2, genotype, array, processingDescription] BINARY: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 68B for [individual2, genotype, array, alleles, array] BINARY: 60 values, 37B raw, 37B comp, 1 pages, encodings: [RLE, PLAIN_DICTIONARY], dic { 2 entries, 14B raw, 2B comp}
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual2, genotype, array, expectedAlleleDosage] FLOAT: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual2, genotype, array, referenceReadDepth] INT32: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual2, genotype, array, alternateReadDepth] INT32: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual2, genotype, array, readDepth] INT32: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual2, genotype, array, minReadDepth] INT32: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual2, genotype, array, genotypeQuality] INT32: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 36B for [individual2, genotype, array, genotypeLikelihoods, array] INT32: 30 values, 19B raw, 19B comp, 1 pages, encodings: [PLAIN, RLE]
```

```
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 36B for [individual2, genotype, array, nonReferenceLikelihoods, array] INT32: 30 values, 19B raw, 19B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 36B for [individual2, genotype, array, strandBiasComponents, array] INT32: 30 values, 19B raw, 19B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual2, genotype, array, splitFromMultiAllelic] BOOLEAN: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual2, genotype, array, isPhased] BOOLEAN: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual2, genotype, array, phaseSetId] INT32: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 32B for [individual2, genotype, array, phaseQuality] INT32: 30 values, 15B raw, 15B comp, 1 pages, encodings: [PLAIN, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 72B for [individual2, genotype, array, allelesBase, array] BINARY: 60 values, 45B raw, 45B comp, 1 pages, encodings: [RLE, PLAIN_DICTIONARY], dic { 4 entries, 20B raw, 4B comp}
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 37B for [relationType] BINARY: 15 values, 8B raw, 8B comp, 1 pages, encodings: [BIT_PACKED, RLE, PLAIN_DICTIONARY], dic { 1 entries, 6B raw, 1B comp}
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 49B for [ibdExpectedValue] DOUBLE: 15 values, 8B raw, 8B comp, 1 pages, encodings: [BIT_PACKED, RLE, PLAIN_DICTIONARY], dic { 1 entries, 8B raw, 1B comp}
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 23B for [z0] DOUBLE: 15 values, 6B raw, 6B comp, 1 pages, encodings: [PLAIN, BIT_PACKED, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 23B for [z1] DOUBLE: 15 values, 6B raw, 6B comp, 1 pages, encodings: [PLAIN, BIT_PACKED, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 23B for [z2] DOUBLE: 15 values, 6B raw, 6B comp, 1 pages, encodings: [PLAIN, BIT_PACKED, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 23B for [ibdProportion] DOUBLE: 15 values, 6B raw, 6B comp, 1 pages, encodings: [PLAIN, BIT_PACKED, RLE]
```

```
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 23B for [pairwisePhenotypicCode] INT32: 15 values, 6B raw, 6B comp, 1 pages, encodings: [PLAIN, BIT_PACKED, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 23B for [ibsDistance] DOUBLE: 15 values, 6B raw, 6B comp, 1 pages, encodings: [PLAIN, BIT_PACKED, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 23B for [pValueBinomialTest] DOUBLE: 15 values, 6B raw, 6B comp, 1 pages, encodings: [PLAIN, BIT_PACKED, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 23B for [ratioIBS0] DOUBLE: 15 values, 6B raw, 6B comp, 1 pages, encodings: [PLAIN, BIT_PACKED, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 23B for [numberNonMissingVariant0] INT32: 15 values, 6B raw, 6B comp, 1 pages, encodings: [PLAIN, BIT_PACKED, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 23B for [numberNonMissingVariant1] INT32: 15 values, 6B raw, 6B comp, 1 pages, encodings: [PLAIN, BIT_PACKED, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 23B for [numberNonMissingVariant2] INT32: 15 values, 6B raw, 6B comp, 1 pages, encodings: [PLAIN, BIT_PACKED, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 23B for [number0SNPInPPCTest] DOUBLE: 15 values, 6B raw, 6B comp, 1 pages, encodings: [PLAIN, BIT_PACKED, RLE]
4-Nov-2015 5:24:06 PM INFO: parquet.hadoop.ColumnChunkPageWriteStore: written 23B for [number2SNPInPPCTest] DOUBLE: 15 values, 6B raw, 6B comp, 1 pages, encodings: [PLAIN, BIT_PACKED, RLE]
```

Log exécution --show-parquet sur le fichier obtenu par --genome (fichier show.testGenome) :

```
root
 |-- individual1: struct (nullable = true)
 |    |-- familyId: string (nullable = true)
 |    |-- individualId: string (nullable = true)
 |    |-- paternalId: string (nullable = true)
 |    |-- maternalId: string (nullable = true)
 |    |-- sex: string (nullable = true)
 |    |-- phenotype: struct (nullable = true)
 |    |    |-- phenotype: string (nullable = true)
 |    |-- genotype: array (nullable = true)
 |    |    |-- element: struct (containsNull = false)
```

```
|    |    |    |-- variant: struct (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)
|    |    |    |    |-- 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)
|    |    |    |    |-- variantCallErrorProbability: float (nullable = true)
|    |    |    |    |-- variantIsPassing: boolean (nullable = true)
|    |    |    |    |-- variantFilters: array (nullable = false)
|    |    |    |    |    |-- element: string (containsNull = false)
|    |    |    |    |-- readDepth: integer (nullable = true)
|    |    |    |    |-- downsampled: boolean (nullable = true)
|    |    |    |    |-- baseQRankSum: float (nullable = true)
|    |    |    |    |-- clippingRankSum: float (nullable = true)
|    |    |    |    |-- fisherStrandBiasPValue: float (nullable = true)
|    |    |    |    |-- haplotypeScore: float (nullable = true)
|    |    |    |    |-- inbreedingCoefficient: float (nullable = true)
|    |    |    |    |-- rmsMapQ: float (nullable = true)
|    |    |    |    |-- mapq0Reads: integer (nullable = true)
|    |    |    |    |-- mqRankSum: float (nullable = true)
|    |    |    |    |-- variantQualityByDepth: float (nullable = true)
|    |    |    |    |-- readPositionRankSum: float (nullable = true)
|    |    |    |    |-- genotypePriors: array (nullable = false)
|    |    |    |    |    |-- element: integer (containsNull = false)
|    |    |    |    |-- genotypePosteriors: array (nullable = false)
|    |    |    |    |    |-- element: integer (containsNull = false)
|    |    |    |    |-- vqslod: float (nullable = true)
```

```
 |    |    |    |    |-- culprit: string (nullable = true)
 |    |    |    |    |-- usedForNegativeTrainingSet: boolean (nullable = true)
 |    |    |    |    |-- usedForPositiveTrainingSet: boolean (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 = true)
 |    |    |    |    |-- 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 = true)
 |    |    |    |    |-- element: integer (containsNull = false)
 |    |    |    |-- nonReferenceLikelihoods: array (nullable = true)
 |    |    |    |    |-- element: integer (containsNull = false)
 |    |    |    |-- strandBiasComponents: array (nullable = true)
 |    |    |    |    |-- element: integer (containsNull = false)
 |    |    |    |-- splitFromMultiAllelic: boolean (nullable = true)
 |    |    |    |-- isPhased: boolean (nullable = true)
 |    |    |    |-- phaseSetId: integer (nullable = true)
 |    |    |    |-- phaseQuality: integer (nullable = true)
 |    |    |    |-- allelesBase: array (nullable = true)
 |    |    |    |    |-- element: string (containsNull = false)
 |-- individual2: struct (nullable = true)
 |    |-- familyId: string (nullable = true)
 |    |-- individualId: string (nullable = true)
 |    |-- paternalId: string (nullable = true)
 |    |-- maternalId: string (nullable = true)
 |    |-- sex: string (nullable = true)
 |    |-- phenotype: struct (nullable = true)
 |    |    |-- phenotype: string (nullable = true)
 |    |-- genotype: array (nullable = true)
 |    |    |-- element: struct (containsNull = false)
 |    |    |    |-- variant: struct (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)
|    |    |    |    |-- 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)
|    |    |    |    |-- variantCallErrorProbability: float (nullable = true)
|    |    |    |    |-- variantIsPassing: boolean (nullable = true)
|    |    |    |    |-- variantFilters: array (nullable = false)
|    |    |    |    |    |-- element: string (containsNull = false)
|    |    |    |    |-- readDepth: integer (nullable = true)
|    |    |    |    |-- downsampled: boolean (nullable = true)
|    |    |    |    |-- baseQRankSum: float (nullable = true)
|    |    |    |    |-- clippingRankSum: float (nullable = true)
|    |    |    |    |-- fisherStrandBiasPValue: float (nullable = true)
|    |    |    |    |-- haplotypeScore: float (nullable = true)
|    |    |    |    |-- inbreedingCoefficient: float (nullable = true)
|    |    |    |    |-- rmsMapQ: float (nullable = true)
|    |    |    |    |-- mapq0Reads: integer (nullable = true)
|    |    |    |    |-- mqRankSum: float (nullable = true)
|    |    |    |    |-- variantQualityByDepth: float (nullable = true)
|    |    |    |    |-- readPositionRankSum: float (nullable = true)
|    |    |    |    |-- genotypePriors: array (nullable = false)
|    |    |    |    |    |-- element: integer (containsNull = false)
|    |    |    |    |-- genotypePosteriors: array (nullable = false)
|    |    |    |    |    |-- element: integer (containsNull = false)
|    |    |    |    |-- vqslod: float (nullable = true)
|    |    |    |    |-- culprit: string (nullable = true)
|    |    |    |    |-- usedForNegativeTrainingSet: boolean (nullable = true)
```

```
 |    |    |    |    |-- usedForPositiveTrainingSet: boolean (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 = true)
 |    |    |    |    |-- 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 = true)
 |    |    |    |    |-- element: integer (containsNull = false)
 |    |    |    |-- nonReferenceLikelihoods: array (nullable = true)
 |    |    |    |    |-- element: integer (containsNull = false)
 |    |    |    |-- strandBiasComponents: array (nullable = true)
 |    |    |    |    |-- element: integer (containsNull = false)
 |    |    |    |-- splitFromMultiAllelic: boolean (nullable = true)
 |    |    |    |-- isPhased: boolean (nullable = true)
 |    |    |    |-- phaseSetId: integer (nullable = true)
 |    |    |    |-- phaseQuality: integer (nullable = true)
 |    |    |    |-- allelesBase: array (nullable = true)
 |    |    |    |    |-- element: string (containsNull = false)
 |-- relationType: string (nullable = true)
 |-- ibdExpectedValue: double (nullable = true)
 |-- z0: double (nullable = true)
 |-- z1: double (nullable = true)
 |-- z2: double (nullable = true)
 |-- ibdProportion: double (nullable = true)
 |-- pairwisePhenotypicCode: integer (nullable = true)
 |-- ibsDistance: double (nullable = true)
 |-- pValueBinomialTest: double (nullable = true)
 |-- ratioIBS0: double (nullable = true)
 |-- numberNonMissingVariant0: integer (nullable = true)
 |-- numberNonMissingVariant1: integer (nullable = true)
 |-- numberNonMissingVariant2: integer (nullable = true)
 |-- number0SNPInPPCTest: double (nullable = true)
```

```
 |-- number2SNPInPPCTest: double (nullable = true)

+--------------------+--------------------+------------+----------------+----+----
+----+-------------+----------------------+-----------+------------------+---------
+------------------------+------------------------+------------------------+-------
------------+--------------------+
|         individual1|         individual2|relationType|ibdExpectedValue|  z0|  z1|
z2|ibdProportion|pairwisePhenotypicCode|ibsDistance|pValueBinomialTest|ratioIBS0|nu
mberNonMissingVariant0|numberNonMissingVariant1|numberNonMissingVariant2|number0SNP
InPPCTest|number2SNPInPPCTest|
+--------------------+--------------------+------------+----------------+----+----
+----+-------------+----------------------+-----------+------------------+---------
+------------------------+------------------------+------------------------+-------
------------+--------------------+
|[1,1,0,0,Male,[1]...|[2,1,0,0,Male,[1]...|                                  UN|
0.0|null|null|null|             null|                            null|          null|
null|       null|                               null|                            null|
null|               null|               null|
|[1,1,0,0,Male,[1]...|[3,1,0,0,Male,[1]...|                                  UN|
0.0|null|null|null|             null|                            null|          null|
null|       null|                               null|                            null|
null|               null|               null|
|[1,1,0,0,Male,[1]...|[4,1,0,0,Male,[2]...|                                  UN|
0.0|null|null|null|             null|                            null|          null|
null|       null|                               null|                            null|
null|               null|               null|
|[1,1,0,0,Male,[1]...|[5,1,0,0,Male,[2]...|                                  UN|
0.0|null|null|null|             null|                            null|          null|
null|       null|                               null|                            null|
null|               null|               null|
|[1,1,0,0,Male,[1]...|[6,1,0,0,Male,[2]...|                                  UN|
0.0|null|null|null|             null|                            null|          null|
null|       null|                               null|                            null|
null|               null|               null|
|[2,1,0,0,Male,[1]...|[3,1,0,0,Male,[1]...|                                  UN|
0.0|null|null|null|             null|                            null|          null|
null|       null|                               null|                            null|
null|               null|               null|
|[2,1,0,0,Male,[1]...|[4,1,0,0,Male,[2]...|                                  UN|
0.0|null|null|null|             null|                            null|          null|
```

```
null|       null|                              null|                              null|
null|               null|               null|
|[2,1,0,0,Male,[1]...|[5,1,0,0,Male,[2]...|                                        UN|
0.0|null|null|null|             null|                           null|          null|
null|       null|                              null|                              null|
null|               null|               null|
|[2,1,0,0,Male,[1]...|[6,1,0,0,Male,[2]...|                                        UN|
0.0|null|null|null|             null|                           null|          null|
null|       null|                              null|                              null|
null|               null|               null|
|[3,1,0,0,Male,[1]...|[4,1,0,0,Male,[2]...|                                        UN|
0.0|null|null|null|             null|                           null|          null|
null|       null|                              null|                              null|
null|               null|               null|
+--------------------+--------------------+------------+----------------+----+----
+----+-------------+----------------------+-----------+------------------+---------
+------------------------+------------------------+-------------------------+-------
------------+-------------------+

4-Nov-2015 5:29:35 PM INFO: parquet.hadoop.ParquetFileReader: Initiating action
with parallelism: 5
4-Nov-2015 5:29:36 PM WARNING: parquet.hadoop.ParquetRecordReader: Can not
initialize counter due to context is not a instance of TaskInputOutputContext, but
is org.apache.hadoop.mapreduce.task.TaskAttemptContextImpl
4-Nov-2015 5:29:36 PM INFO: parquet.hadoop.InternalParquetRecordReader:
RecordReader initialized will read a total of 15 records.
4-Nov-2015 5:29:36 PM INFO: parquet.hadoop.InternalParquetRecordReader: at row 0.
reading next block
4-Nov-2015 5:29:36 PM INFO: parquet.hadoop.InternalParquetRecordReader: block read
in memory in 19 ms. row count = 15
```

ANNEXE VI

Extension du modèle adam-ibs pour les besoins en analyse génotypiques des études cliniques.

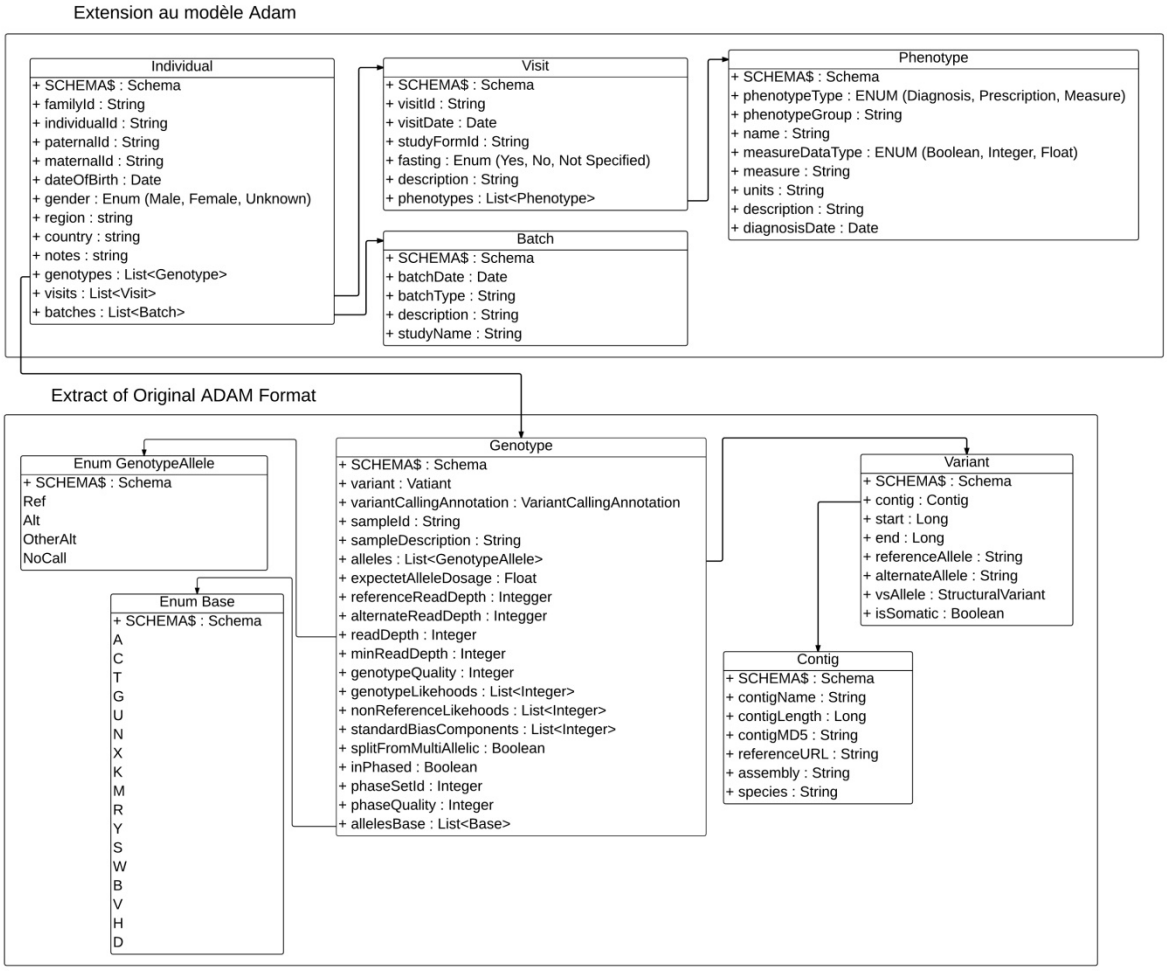

Figure 13 : Modèle de données intégrant les besoins de CRCHUM pour l'analyse des études cliniques

BIBLIOGRAPHIE

[1] Vivien Marx, "Genomics in the clouds", Nature methods, Vol.10 No.10, Octobre 2013.

[2] Aisling O'Driscoll, Jurate Daugelaite, Roy D. Sleator, "Big data, Hadoop and cloud computing in genomics", Elsevier, Journal of Biomedical Informatics, 2013.

[3] Krampis K, Booth T, Chapman B, Tiwari B, Bicak M, Field D, et al.. CloudBioLinux: pre-configured and on-demand bioinformatics computing for the genomics community, BMC Bioinform, 2012.

[4] Uday S Evani, Danny Challis, Jin Yu, Andrew R Jackson, Sameer Paithankar, Matthew N Bainbridge, Adinarayana Jakkamsetti, Peter Pham, Cristian Coarfa, Aleksandar Milosavljevic, Fuli Yu, "Atlas2 Cloud: a framework for personal genome analysis in the cloud", BMC Genomics, 2012.

[5] Galaxy Project, https://galaxyproject.org, consulté le 05/09/2015.

[6] Bo Liu, Ravi K Madduri, Borja Sotomayor, Kyle Chard, Lukasz Lacinski, Utpal J Dave, Jianqiang Li, Chunchen Liu, Ian T Foster, "Cloud-based bioinformatics workflow platform for large-scale next-generation sequencing analyses", Journal of Biomedical Informatics, 2014.

[7] Globus Genomics, Computation Institute, University of Chicago, Argonne National Laboratory, https://www.globus.org/genomics, consulté le 05/09/2015.

[8] Krithika Bhuvaneshwar, Dinanath Sulakhe, Robinder Gauba, Alex Rodriguez, Ravi Madduri, Utpal Dave, Lukasz Lacinski, Ian Foster, Yuriy Gusev, Subha Madhavan, "A case study for cloud based high throughput analysis of NGS data using the globus

genomics system", Elsevier, Computational and Structural Biotechnology Journal, 2014.

[9] Matt Massie, Frank Nothaft, Christopher Hartl, Christos Kozanitis, André Schumacher, Anthony D. Joseph and David A. Patterson, "ADAM: Genomics Formats and Processing Patterns for Cloud Scale Computing", University of California, Berkeley, 2013.

[10] Matei Zaharia, Ankur Dave, Michael J. Franklin, Mosharaf Chowdhury, Justin Ma, Scott Shenker, Tathagata Das, Murphy McCauley, Ion Stoica, "Fast and Interactive Analytics over Hadoop Data with Spark", UC Berkeley, 2012.

[11] Apache Hadoop, http://hadoop.apache.org, consulté le 03/08/2015.

[12] Apache Avro, https://avro.apache.org, consulté le 03/08/2015.

[13] Apache Parquet, https://parquet.apache.org, consulté le 03/08/2015.

[14] Sequencing Whole Genomes, http://www.bio.davidson.edu/courses/genomics/method/shotgun.html, consulté le 03/11/2015.

[15] Évolution des méthodes de génotypage, Matthieu FALQUE, UMR de Génétique Végétale, INRA − Univ Paris-Sud − CNRS − AgroParisTech, Ferme du Moulon, F-91190 Gif-sur-Yvette, France.

[16] Genome-Wide Association Studies, https://www.genome.gov/20019523, consulté le 03/11/2015.

[17] Purcell S1, Neale B, Todd-Brown K, Thomas L, Ferreira MA, Bender D, Maller J, Sklar P, de Bakker PI, Daly MJ, Sham PC., "PLINK: a tool set for whole-genome

association and population-based linkage analyses", The American Journal of Human Genetics. 2007.

[18] Spark Cluster Mode Overview, https://spark.apache.org/docs/latest/cluster-overview.html, consulté le 20/08/2015.

[19] Understanding how Parquet integrates with Avro, Thrift and Protocol Buffers, http://grepalex.com/2014/05/13/parquet-file-format-and-object-model, consulté le 10/09/2015.

[20] Model de donnés bgd-formats d'Adam, https://github.com/bigdatagenomics/bdg-formats, consulté le 28/09/2015.

[21] Outil PLink, https://www.cog-genomics.org/plink2, consulté le 28/09/2015.

[22] Projet Adam-Ibs algorithme de calcul Pairwise_IBS_IBD, https://github.com/GELOG/adam-ibs/wiki/Algorithm-for-Pairwise--IBS-IBD-computation-%28--genome%29, consulté le 28/09/2015.

[23] Projet Logback, http://logback.qos.ch/index.html, consulté le 28/09/2015.

[24] Projet Variantminer, https://github.com/GELOG/variantminer, consulté le 01/11/2015.

[25] Ralph Kimball, Joe Caserta, « Chapter 1: Surrounding the Requirements » dans The Data Warehouse ETL Toolkit. Wiley Publishing, Inc. 2004.