ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA EN SISTEMAS

DESARROLLO Y EVALUACIÓN DE RENDIMIENTO DE UNA APLICACIÓN PARA EL ANÁLISIS DE ADN UTILIZANDO HADOOP

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERÍA EN SISTEMAS INFORMÁTICOS Y DE COMPUTACIÓN

JUAN CARLOS PAUCAR LÓPEZ

juan.paucar@outlook.com

DIRECTOR: ING. IVÁN CARRERA, MSc.

ivan.carrera@epn.edu.ec

Quito, junio 2018

DECLARACIÓN

Yo, Juan Carlos Paucar López, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentada para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedo mis derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

Juan Carlos Paucar López

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por Juan Carlos Paucar López, bajo mi supervisión.

Ing. Iván Carrera, MSc.
Director del Proyecto

AGRADECIMIENTOS

A mi familia, en especial a mi padre por esforzarse siempre por que no nos falten nada a mi hermana y a mí. Él hizo posible todo este viaje con su esfuerzo y dándome cada posibilidad que desembocó en mi fascinación por la computación.

También quisiera agradecer a mis maestros durante la carrera. Por siempre buscar maneras de que aprendiera. Especialmente a mi director por todo el tiempo invertido.

Finalmente, a todas las personas que desinteresadamente, y sin tener la obligación, me enseñaron sobre temas que cada día siguen fascinándome con nuevas aplicaciones en la computación. Por respetar siempre mi opinión y responder mis dudas.

DEDICATORIA

Este trabajo de titulación está dedicado a mi familia por su apoyo durante todo este tiempo.

También está dedicado a mis abuelitos, que partieron durante mis años en la Universidad.

CONTENIDO

LISTADO DE TABLAS

LISTADO DE FIGURAS

LISTADO DE FRAGMENTOS DE CÓDIGO

RESUMEN

Durante las últimas dos décadas, desde el primer secuenciamiento del genoma humano, la cantidad de información que genera este proceso ha sido considerablemente grande. El ritmo con el que se genera los datos ha estado siempre por delante de la capacidad contemporánea de procesamiento. El manejo de esta gran cantidad de datos presenta nuevos retos. Estos retos requieren el desarrollo de nuevas herramientas y algoritmos capaces de procesar, de manera eficiente dichos datos.

El presente proyecto de titulación trata acerca del desarrollo de una aplicación para el procesamiento de una fase en el análisis genético; posteriormente, evaluar el desempeño de la aplicación para generar un modelo predictivo de rendimiento, para poder predecir el tiempo de ejecución de la aplicación. Dicha aplicación hace uso de un framework de MapReduce como Hadoop; así también, hace uso un paradigma de programación diferente a la Programación Orientada a Objetos, como la Programación Funcional. El uso de ambos se justifica como apropiados para el procesamiento de datos.

Palabras claves – Bioinformática, VCF, MapReduce, Hadoop, Programación Funcional, Haskell, Evaluación de desempeño

ABSTRACT

During the last two decades, since the first sequencing of the human genome, the amount of information generated by this process has been quite large. The pace at which data is generated has always been ahead of contemporary processing capacity. Handling this large amount of data presents new challenges. These challenges require the development of new tools and capable of processing all that data efficiently.

This work deals with the development of an application for the processing of a phase in genetic analysis; subsequently, evaluate the performance of the application is to generate a predictive model of performance to be able to predict the execution time of the application under different circumstances. This application uses a MapReduce framework like Hadoop; it also uses a paradigm of programming different from the Object-Oriented Programming like Functional Programming. The use of both is justified as appropriate for data processing.

Keywords – Bioinformatics, VCF, MapReduce, Hadoop, Functional Programming, Haskell, Performance Evaluation

CAPÍTULO 1. BASE CONCEPTUAL

1.1 OBJETIVOS

1.1.1 OBJETIVO GENERAL

Realizar el desarrollo y la evaluación del desempeño computacional de una aplicación para el análisis de ADN que utilice el framework Hadoop.

1.1.2 OBJETIVOS ESPECÍFICOS

- Describir un algoritmo para la identificación de efectos de las variaciones genéticas basado en el procesamiento de archivos VCF.
- Desarrollar una aplicación distribuida para análisis de los efectos de variaciones genéticas anotadas en archivos VCF.
- Generar un modelo predictivo de rendimiento para la aplicación en función de la capacidad de procesamiento y número de servidores.

1.2 MOTIVACIÓN

El análisis de ADN es un conjunto de procedimientos que cada vez son más fáciles de conseguir desde el primer secuenciamiento del genoma humano[8]. El secuenciamiento del ADN tiene diversas aplicaciones en varios campos[18], por ejemplo:

- Biología Molecular
- Biología Evolutiva
- Metagenómica
- Medicina

En el caso de la medicina, las pruebas en el ADN del paciente pueden dar lugar al hallazgo de enfermedades congénitas[21]. Debido a que es el núcleo de la célula guarda la mayor parte del material genético de un individuo[32], contiene información acerca de las enfermedades que podría desarrollar a lo largo de su vida y que pudo heredar de sus progenitores[21], o cuya herencia lo vuelva más propenso a desarrollar alguna condición o enfermedad.

Los exámenes de ADN son costosos debido a que necesitan de un hardware de gran capacidad para realizar los análisis en el menor tiempo posible, y sistemas embebidos con software propio para el análisis del ADN[40]. El descubrimiento temprano de enfermedades congénitas, puede ayudar a un tratamiento en etapas tempranas de ciertas enfermedades e incluso a la prevención de las mismas[21]. Varios trabajos que se realizaron en los últimos años, han logrado abaratar los costos de estos análisis y que sean más asequibles a la mayoría de personas [40]. El descenso en el costo como vemos en la figura 1.1; la figura tiene el eje Y, representando el costo en una escala logarítmica, que nos presenta una idea del drástico descenso del costo.

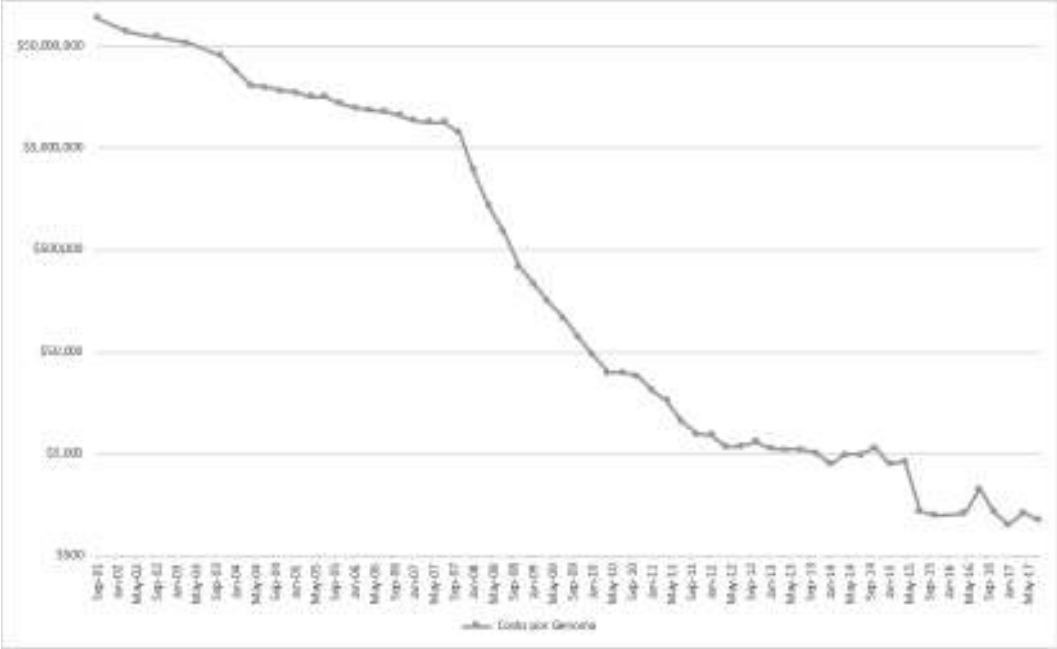

Figura 1.1: Costos del secuenciamiento del genoma humano[40]

Existe un conjunto de aplicaciones de bioinformática que han sido desarrolladas para poder ayudar con las diversas etapas análisis de ADN [35]. Durante todas las etapas del análisis de ADN, se requiere un alto desempeño computacional, siendo especialmente intensivos en uso de CPU y lecturas en disco [35]. Esto se debe a que en las primeras etapas los archivos que se manejan son de hasta cientos de giga bytes de tamaño [26] y deben ser cotejados con el genoma humano completo [20].

En la actualidad, podemos encontrar ambientes con un alto desempeño computacional y costos decrecientes en la nube [35]. Esto causa que sea atractivo el uso del poder computacional de la nube para las aplicaciones de bioinformática [35]. Dado que las aplicaciones de bioinformática son intensivas en procesamiento y almacenamiento, las aplicaciones todavía deben hacer un uso correcto de recursos [36]. La configuración de los equipos también juega un papel fundamental, esta configuración debe tener consideraciones adecuadas a ambientes distribuidos en la nube [36].

El presente proyecto beneficiará a centros de estudios estudios genéticos a indivi-

duos que deseen usar la aplicación para la búsqueda de efectos de una variante en los archivos VCF. El proceso de búsqueda de patologías parte de la toma de la muestra de ADN de un paciente, y esta involucra varias fases como: el secuenciamiento, el alineamiento, anotación y priorización de variaciones [4]. Cada fase presenta sus propios retos técnicos en diversos campos [14]. El presente proyecto apunta al desarrollo de un software, para ofrecer un reporte formal acerca de las patologías y enfermedades congénitas halladas tras una búsqueda sobre las variaciones encontradas y su relación con las patologías.

La búsqueda de las patologías es un proceso con un costo computacional alto [35], debido a que involucra el procesamiento de cantidades masivas de datos generados tras los procesos de secuenciamiento y alineación [20], que tras la anotación de variaciones tiene gran tamaño [4]. Sin embargo, mientras más se logre optimizar su procesamiento, ya sea mediante nuevos algoritmos para el procesamiento o usando nuevas técnicas para procesamiento paralelo de masivas cantidades de datos, la búsqueda de variaciones y patologías que estás podrían involucrar, sería mucho más rápida. Esto haría que el costo por análisis de las variantes en los genes sea menor y que tome menos tiempo. Con un costo menor, el número de pacientes que se podrían beneficiar es mayor [35]. Y, finalmente, un análisis que tome menos tiempo significa un menor tiempo entre el diagnóstico y un tratamiento o medidas preventivas para el paciente.

1.3 CONTRIBUCIÓN

El presente proyecto de titulación inicia con una investigación sobre el estado del arte de las aplicaciones distribuidas de análisis de ADN. Se selecciona y describe un modelo de aplicación para el análisis de variantes encontradas en el ADN en ambientes distribuidos usando Hadoop. Una vez seleccionado el modelo, se procede a realizar la implementación utilizando un lenguaje de programación funcional. Se identifican también las características funcionales y no funcionales del modelo de aplicación, que tengan un efecto sobre el desempeño computacional a través de un

análisis de la aplicación. El uso de un lenguaje de programación con un paradigma diferente al de la orientación a objetos, permitirá dar un enfoque novedoso acerca de cómo estas características afectan el análisis de grandes cantidades de datos tanto de manera distribuida como en paralelo. Así también, el uso de frameworks para el procesamiento distribuido de datos como Hadoop y cómo los conceptos básicos detrás de su funcionamiento se pueden aplicar para construir soluciones propias que requieran agilizar la configuración. Finalmente, se evalua el desempeño computacional de la aplicación implementada en un ambiente de pruebas, en donde se utiliza una metodología de benchmarking, comparando la implementación en diferentes ambientes de ejecución. Todo esto permite un análisis a fondo de los componentes dentro de esta clase de aplicaciones que se pueden mejorar a futuro.

1.3.1 HADOOP Y HASKELL

Tal como se mencionó en el apartado 1.2, existe una gran cantidad de herramientas que se utilizan en las diferentes fases del análisis de ADN. En el presente proyecto se pretende aportar a este ecosistema con el uso de lenguajes funcionales. Debido a que Hadoop viene junto con Hadoop Streaming[1], muchos usuarios de Hadoop han podido dejar de depender de Java, el lenguaje por defecto para el uso de Hadoop[12]. Esto permite a diversos investigadores, de varias áreas, sortear la barrera de aprendizaje de un lenguaje de programación específico para el uso de una herramienta que permite el procesamiento masivo y paralelo de grandes cantidades de datos. En el caso del presente proyecto, esto se traduce en la posibilidad de aprovechar las ventajas que nos puede ofrecer la programación funcional junto con los lenguajes funcionales. Se abordarán conceptos básicos en la programación funcional así como la Teoría de Categorías que, en lenguajes funcionales como Haskell, buscan darle una base teórica con propiedades matemáticas sobre la cual hacer desarrollo de software.

El lenguaje escogido para para la implementación es Haskell, un lenguaje con una gran comunidad, académica y profesional, que lo respalda con una gran cantidad

[1] Sitio: https://wiki.apache.org/hadoop/HadoopStreaming

de librerías en su haber[2], así como también una gran cantidad de literatura[3] e investigación[4]. Haskell también cuenta con librerías propias apropiadas para el presente proyecto y nos entrega una aplicación que contiene los binarios necesarios para Hadoop, así como también toma a cargo los parámetros que Hadoop Streaming debe recibir. El uso de un lenguaje funcional, con una sintaxis declarativa, como Haskell [17] significa un software eficiente [38] y más fácilmente extensible en el tiempo, sin llegar a sacrificar el rendimiento, incluso a lenguajes como C [27]. Además, existe la posibilidad de usar Hadoop para analizar la gran cantidad de datos que contienen los archivos con las variaciones encontradas en los genes.

1.3.2 PRUEBAS DE RENDIMIENTO

En el presente proyecto se espera realizar un modelo de rendimiento de la aplicación que permite predecir cuál será el tiempo que tome la ejecución de la aplicación [5], bajo diferentes entornos. Para la generación de este modelo se realiza pruebas de rendimiento se hará siguiendo un acercamiento sistemático para la evaluación de rendimiento [22] de manera que al final del proyecto no se obtenga solamente un software de gran utilidad para centros de salud, sino que se tendrá información de relevancia acerca del rendimiento obtenido. Esta información es de importancia para la comparación con soluciones existentes y trabajos futuros. La información del rendimiento también es útil para el creciente y gran ecosistema de aplicaciones relacionada a la genética que se tiene actualmente, mismo que puede tomar como punto de partida la contribución que este proyecto de titulación para el uso de algunos de los conceptos presentados aquí.

[2] Repositorio central de paquetes y librerías: http://hackage.haskell.org/

[3] Libros relacionados a Haskell: https://wiki.haskell.org/Books

[4] Recursos para el aprendizaje de Haskell: https://wiki.haskell.org/Learning_Haskell

1.4 ORGANIZACIÓN

El presente proyecto de titulación está organizado en cuatro capítulos que abarcan desde la base conceptual junto con las razones y motivaciones del proyecto, hasta el análisis de rendimiento de la solución implementada. Finalmente, se realizar conclusiones y recomendaciones.

El primer capítulo contiene todo lo relacionado con establecer un marco de referencia básico acerca de la motivación para la realización del presente proyecto de tesis, así como las contribuciones resultado de la realización del proyecto de tesis.

En el segundo capítulo se detalla cómo se consiguen y cuáles son los requerimientos necesarios en la aplicación a desarrollar, y se explica cuál es la parte exacta en el análisis de ADN, en que se enfocará la aplicación. De esta manera, se podrá comparar con las diferentes soluciones ya desarrolladas que existen para el problema dentro del análisis de ADN, de manera que se pueda usar las aplicaciones ya desarrolladas como una referencia de la cual podremos extraer puntos fuertes y débiles. Finalmente, en este capítulo se justifica las herramientas que se eligieron para realizar los requerimientos de la aplicación como son Hadoop y Haskell, dando un análisis técnico de las mismas.

En el tercer capítulo se detalla la implementación de la aplicación. Se comienza por definir las características de los ambientes donde la aplicación se ejecutará. Con los requisitos definidos en capítulos anteriores, se procederá a realizar la implementación de la aplicación. Para que sea una aplicación posteriormente mantenible, se realiza tanto pruebas unitarias como de aceptación, debido a que, si bien el sistema de tipos de Haskell nos provee una manera más segura de escribir código asegurando la lógica de los tipos [17], siempre es una buena idea contar con un respaldo acerca del comportamiento de la funciones dentro del programa. Finalmente, se describirán los problemas que pudieron haberse encontrado durante el desarrollo, a manera de registro para trabajos posteriores.

En el cuarto capítulo, y una vez terminada la implementación de la aplicación, se

describen las pruebas del rendimiento computacional del sistema. Primero, se realiza un Plan de Pruebas que contendrá toda la información relacionada a las pruebas que se harán para medir el rendimiento de la aplicación. Posteriormente, se realiza un análisis de los resultados obtenidos durante las pruebas para generar un modelo de predicción del tiempo de ejecución.

Tras el análisis de los requisitos, implementación de la aplicación y realización de un modelo de predicción de rendimiento de la aplicación. Se describen las conclusiones a las que se llegó al realizar el presente proyecto de tesis así como las recomendaciones para trabajos futuros en la misma línea o afines.

CAPÍTULO 2. DISEÑO

2.1 REQUERIMIENTOS DE LA APLICACIÓN

2.1.1 DETALLE DE LA APLICACIÓN

La aplicación a desarrollar realizará la búsqueda de los efectos que las variaciones, dentro del ADN, podrían a tener en pacientes. La aplicación deberá funcionar en conjunto con el *framework* Hadoop para el procesamiento de los datos en paralelo de manera distribuida.

Este análisis corresponde a la última fase del proceso que comienza con la extracción de una muestra de ADN del paciente, para posteriormente ser secuenciada y llevada a través de una serie de procesamientos de los datos encontrados [20], hasta que finalmente se obtiene un archivo VCF, donde se almacena información acerca de un genotipo o acerca de una determinada posición en el genoma [16].

Los archivos VCF pueden representar también la información acerca de las variaciones encontradas en el ADN de un conjunto de personas [16], por lo que pueden ser de tamaño considerable. La aplicación debe estar en capacidad procesar la información dentro de estos archivos y cotejarla con una base de datos que contenga los efectos que una variación en el ADN puede tener. Se debe indicar que una variación puede no tener un rol directo en una enfermedad al afectar la función de un gen [31]. Es aquí donde viene la base de datos de los efectos registrados de ciertas variaciones. Estas variaciones también están almacenadas dentro de archivos VCF.

Las variaciones en los genes, respecto a los genomas que se tienen como base, es de hecho bastante común [6]. Todos los seres vivos contienen variaciones aleatorias dentro de sus genes. La mayoría de mutaciones no son peligrosas ni tienen un efecto sobre el ser vivo [6]. Estos efectos visibles son llamados el genotipo de un ser vivo. En cierto casos, la selección natural tiende a favorecer más un alelo en particular en ciertas poblaciones [6]. Sin embargo, existen variaciones en los genes que pueden tener un efecto directo en la salud [6].

Además del archivo VCF contiendo las variaciones de un individuo o población, la aplicación recibirá otro archivo VCF en el que se especifica aquellos códigos de variaciones que de hecho tienen un efecto asociado a alguna patología sobre la salud de una persona. Estos efectos, relacionados a variaciones particulares, se encuentran de manera libre en recursos en línea de varias organizaciones dedicadas al estudio genético. La aplicación hará uso de la base de datos de Emsembl [1].

La aplicación debe ser capaz de ejecutarse en un ambiente distribuido para que pueda realizar el análisis paralelo de los archivos VCF y los coteje con las bases de datos de efectos conocidos de ciertas variaciones. Este ambiente distribuido está dado por Hadoop. Hadoop simplifica el procesamiento de datos en paralelo y de manera distribuida, siempre que se siga con las especificaciones que requiere la aplicación que se va a ejecutar [41]. La salida de la aplicación entrega como resultado un informe con las variaciones que pueden resultar en afectaciones a la salud de un paciente o de una población. Esta clases de pruebas son realizadas por laboratorios especializados en genética. Por esto se debe detallar el procedimiento para ejecutar la aplicación en entornos en la nube, así como automatizar y simplificar la instalación y configuración de la aplicación, dentro de la medida que nos permiten herramientas para provisión de servidores.

La arquitectura de Hadoop requiere de la separación de la lógica de la aplicación en dos partes. El modelo de programación de MapReduce consta de un *mapper* y un *reducer* [41]. En la figura 2.1 se describe la arquitectura de alto nivel de la aplicación. La aplicación toma como entra un archivo VCF, dicho archivo es subido

[1] Sitio web de Emsembl: https://www.ensembl.org/index.html

al entorno de Hadoop. La aplicación dentro de Hadoop está dividida en el mapper y el reducer, ambos son binarios resultados de la compilación de la aplicación en Haskell. El mapper corresponde al primer paso dentro de una aplicación MapReduce [41], este es el encargado de procesar los archivos VCF y convertirlos en una salida de llave y valor que serán enviados a los reducer [41]. Los datos llegan a los reducer agrupados por sus llaves, en nuestra aplicación se asociará a las variaciones por su identificación única. Los reducer deberán tomar los datos de bases locales o llamadas a APIs externas para determinar el efecto de una variación. Finalmente se obtendrá los efectos de las variaciones ecnontradas.

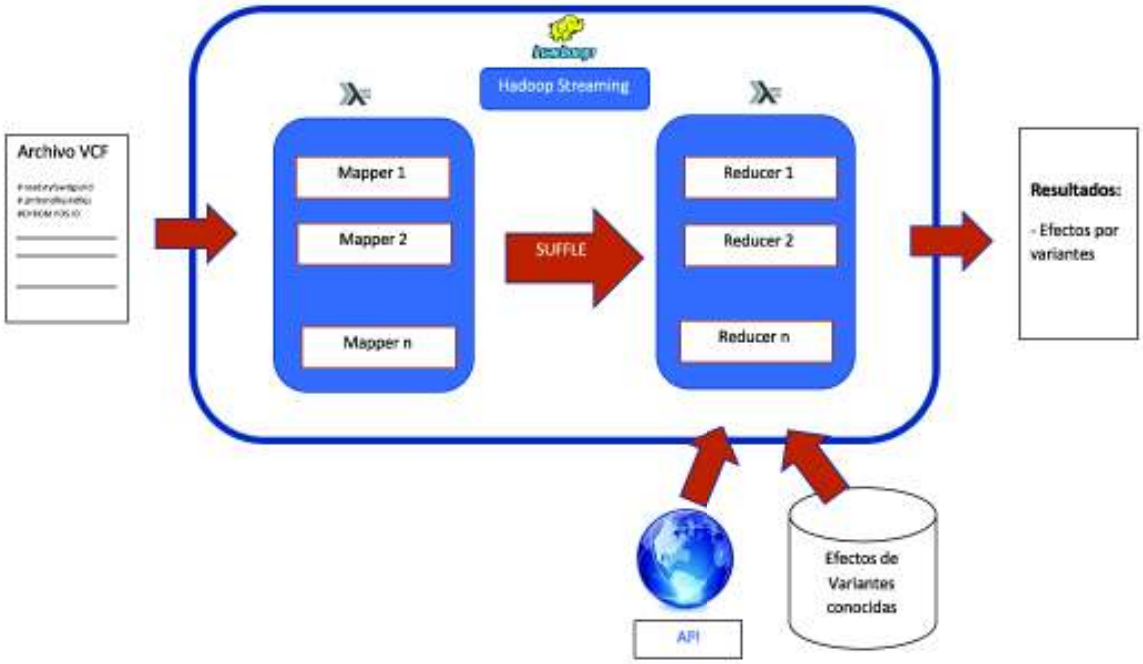

Figura 2.1: Diagrama de la aplicación

2.1.2 REQUERIMIENTOS FUNCIONALES DE LA APLICACIÓN

Un requerimiento funcional describe una función de un componente del sistema [9]. Esta función es descrita como un conjunto de entradas, comportamiento y salidas [9]. En el diagrama de la aplicación, en la figura 2.1, se describe la funcionalidad de alto nivel del sistema. A partir de dicha funcionalidad se describen los siguientes requerimientos funcionales que la aplicación debe cumplir para el propósito descrito de anotación de variantes, son los siguientes:

Tabla 2.1: Requerimientos funcionales de la aplicación

Requerimiento	**Dificultad**	**Importancia**
La aplicación debe poder leer archivos VCF desde la entrada estándar del sistema STDIN	Baja	Alta
La aplicación debe poder escribir a la salida estándar del sistema en sistemas UNIX STDOUT	Baja	Alta
La aplicación debe poder usar las bases de datos de efectos conocidos de variaciones en el ADN, contenidas en archivos VCF	Media	Alta
La aplicación debe devolver un listado con las variaciones cuyos efectos conocidos sean patologías en la salud	Media	Alta
La aplicación debe poder procesar archivos VCF de un paciente o una población	Media	Alta
La aplicación debe funcionar sobre Hadoop para poder ser procesado de manera paralela	Media	Alta
La configuración de la aplicación debe ser compatible para su ejecución con Hadoop Streaming	Alta	Media

2.1.3 REQUERIMIENTOS NO FUNCIONALES DE LA APLICACIÓN

Los requerimientos no funcionales son aquellos que pueden ser usados para validar la operación de un sistema en vez de una función o comportamiento específico [39]. Estos requerimientos no tienen que ver con el diseño del sistema sino con la arquitectura del mismo [39]. En la figura 2.1 podemos ver la arquitectura a alto nivel, la arquitectura ahí descrita está regida por los requerimientos no funcionales de la aplicación. Para la aplicación a desarrollar se ha identificado los siguientes requerimientos.

Tabla 2.2: Requerimientos de la aplicación

Requerimiento	**Dificultad**	**Importancia**
La aplicación debe ser eficiente en el procesamiento de grandes archivos VCF	Media	Alta
La aplicación debe ser realizada siguiendo el paradigma MapReduce para que sea más fácilmente escalable con el uso de Hadoop	Media	Alta
La aplicación debe ser mantenible.	Baja	Media
La aplicación debe poseer pruebas que sirvan de documentación y prevengan errores de regresión en futuros desarrollos	Baja	Media
La aplicación debe ser portable en sistemas operativos UNIX que soporten la entrada estándar del sistema STDIN	Media	Media
La aplicación debe manejar la información contenida en los VCF de una manera segura, manteniendo un tratamiento seguro de los datos	Baja	Alta
La configuración de la aplicación debe estar afinada para su ejecución en la nube	Alta	Media

2.2 COMPARACIÓN CON SOLUCIONES EXISTENTES

El presente proyecto busca realizar una aplicación que pueda determinar las patologías asociadas a determinadas variaciones en los genes como se describe en la sección 2.1.1. El formato VCF fue diseñado, en primer lugar, por 1000 Genomes Project [2], y fue expandido y gestionado por el grupo de trabajo de Alianza Global para la Genómica y Datos de la Salud [3]. Una gran cantidad de organizaciones usan este formato y existen muchas aplicaciones que usan la información contenida dentro de estos archivos. Existen diversas herramientas para la anotación de variantes, algunas muy especificas para ciertos SNPs. Estas pequeñas variaciones en una región muy diminuta del ADN puede manifestarse como una enfermadad o una característica propia de una persona [25]. Para la comparación con soluciones ya existentes, analizaremos brevemente tres aplicaciones que se acercan al objetivo de encontrar efectos o patologías relacionadas a las variaciones en determinados genes. Las aplicaciones a analizar serán:

- AnnoVar
- snpEff
- VEP (Variant Effect Predictor)

2.2.1 ANNOVAR

ANNOVAR es un herramienta, diseñada por el Dr. Kai Wang, que usa información actualizada para la anotación de variaciones genéticas de diversos genomas [1]. Estas variaciones son buscadas a partir de diversos genomas de referencia disponibles, no limitándose a genomas humanos [1]. Esta herramienta es especialmente

[2] What is VCF and should I interpret it? http://gatkforums.broadinstitute.org/gatk/discussion/1268/what-is-a-vcf-and-how-should-i-interpret-it

[3] Global Alliance for Genomics and Health Data Working Group file format team http://ga4gh.org/#/fileformats-team

buena para determinar con exactitud conjuntos de variantes funcionalmente importantes [1]. Además la herramienta usa predicción de mutaciones por anotación, es decir, determina posibles mutaciones en grupos específicos de genes, con funciones específicas, basándose en la información disponible. ANNOVAR está escrita en el lenguaje Perl y puede ser ejecutada como una aplicación autónoma en varios sistemas siempre que tengan módulos estándar de Perl instalados [1].

2.2.2 SNPEFF

SnpEff está descrita como una herramienta para la anotación de variantes y predicción de efectos [7]. El uso típico de esta herramienta consiste en tomar como entrada el resultado de experimentos de secuenciamiento, donde se especifica SNPs, eliminaciones e inserciones de aminoácidos [7]. La salida de este proceso anota las variantes y calcula los efectos de las variantes en genes conocidos como cambios en nucleótidos [7]. SnpEff categoriza rápidamente los efectos de las variantes en las secuencias genómicas [7]. Una vez que el genoma está secuenciado, la aplicación anota las variantes basadas en las posiciones genómicas y predice los efectos de codificación del gen. Las localizaciones genómicas anotadas incluyen intrónicas, regiones no traducidas, *upstreams*, *downstreams*, sitios particionados y regiones intergénicas, entre otros [7].

2.2.3 VEP (VARIANT EFFECT PREDICTOR)

La herramienta VEP (Variant Effect Predictor) fue desarrollada por Ensembl [4]. Esta herramienta permite determinar los efectos de las variantes como SNPs, inserciones, borrados, CNVs o variantes estructurales en genes, transcritos y secuencias de proteínas [10]. Además también tiene las siguientes características que son útiles para estudios que no solamente involucran los efectos de las variantes [10]:

- **Símbolo genético**: Agrega el símbolo genético para la salida del gen. Estos

[4] Sitio de VEP en Ensembl: https://www.ensembl.org/info/docs/tools/vep/index.html

identificadores son establecidos por el HGNC [5].

- **CCDS**: Agrega el identificador transcrito CCDS [6].
- **Proteínas**: Agrega el identificador *Ensembl* de las proteínas.
- **Uniprot**: Agrega identificadores para productos de proteínas desde tres diferentes bases de datos.
- **HGVS**: Genera identificadores HGVS [7] para las entradas relativas a una transcripción de una secuencia (HGVSc) o secuencias de proteínas (HGVSp).
- **Variantes co-localizadas**: Reporta variantes co-localizadas conocidas de la base de datos de variaciones conocidas de Ensembl.

[5] HGNC (HUGO Gene Nomenclature Committee): http://www.genenames.org/
[6] Proyecto CCDS: http://www.ensembl.org/info/genome/genebuild/ccds.html
[7] Human Genome Variation Society: http://varnomen.hgvs.org/

2.2.4 COMPARATIVA ENTRE LAS SOLUCIONES

A continuación se presenta una tabla comparativa con las características de las diferentes herramientas que se especificaron anteriormente [29]; también se hace una comparación con la solución a ser desarrollada en este proyecto. Se pueden notar diferentes características que podrían ser agregadas posteriormente, como una extensión de las capacidades de la aplicación. Los campos a comparar son aquellos que guardan relevancia para la elección de una herramienta u otra para el problema planteado. También se ve la comptaibilidad con **SNP**: variación de un único nuclétido; **INDEL**: inserciones o borrados de nucléotidos y **CNV**: duplicados de nucleótidos base.

Tabla 2.3: Comparación soluciones propuestas

Nombre	**Entradas**	**Salidas**	**SNP**	**INDEL**	**CNV**	**Tipo de Aplicación**
Annovar	VCF, pileup, GFF3-SOLiD, SOAPsnp, MAQ, CASAVA	TXT	Sí	Sí	Sí	CLI
snpEff	VCF, pileup/TXT	VCF, TXT, HTML overview	Sí	Sí	No	CLI
VEP (Variant Effect Predictor)	VCF, pileup, HGVS, TXT, variant identifiers	TXT	Sí	Sí	No	CLI, Web limitado al sitio
Solución propuesta	VCF	TXT, CSV	Sí	Sí	No	CLI

2.3 JUSTIFICACIÓN EN EL USO DE HERRAMIENTAS ESCOGIDAS EN EL DISEÑO

2.3.1 ANÁLISIS DE VARIACIONES EN EL ADN

Como se definió en el primer capítulo, la aplicación que se desarrolla en el presente proyecto de titulación está orientada al análisis de variaciones en el ADN de pacientes. El análisis de ADN es un proceso que involucra varios pasos desde la extracción de la muestra biológica del paciente, hasta poder determinar qué variaciones, halladas en el ADN del paciente, están asociadas con qué efectos o patologías [20]. La aplicación que se desarrollará es la última fase en este proceso, una vez que ya se ha realizado un análisis de las variaciones encontradas en un paciente o en una población.

El primer secuenciamiento de genoma humano, que finalizó en 2001, duró 15 años y costó aproximadamente 3 billones de dólares [20]. Hoy en día, los secuenciadores de últimas generacines pueden secuenciar alrededor de 45 secuencias de genoma humanos por día, con un costo de 1000 dólares cada uno [20]. Esto significa que generamos más datos de los que podemos analizar en el mismo intervalo de tiempo y se espera que los secuenciadores sigan mejorando en velocidad [35]. Esto se traduce en un requerimiento esencial de la aplicación: hacer un software que sea eficiente en el manejo de recursos para que pueda realizar los análisis de una manera veloz.

En la actualidad existen muchas ofertas para el secuenciamiento de los genes de una persona y el costo de estos tiende a bajar de manera exponencial, como se ve en la figura 1.1. Hay empresas que han hecho los análisis de ADN más asequibles como *23andMe* [8], en la que, por 150 dólares se puede hacer un secuenciamiento y anotación de variaciones parcial del ADN. Además, el sitio provee un archivo con la información resultante del secuenciamiento que se puede convertir a archivo VCF. *23andMe* ofrece un análisis parcial de los efectos provocados por variantes

[8] Sitio de 23andMe: https://www.23andme.com

en el genoma. De esta manera, el software como el que se desarrolla en el presente proyecto, puede procesar los datos y encontrar otros efectos no listados por la aplicación.

2.3.2 OPERATIVO PARA HADOOP

Para manejar, de manera eficiente, una cantidad tan grande de datos como los generados por los secuenciadores, podemos recurrir a la paralelización del procesamiento de los mismos. Manejar el procesamiento paralelo de datos involucra una gran complejidad [38], para mejorar el proceso, en el presente proyecto se plantea el uso de Hadoop. El uso de Hadoop hace que la mayor parte de la lógica necesaria para el procesamiento simultáneo de datos sea manejado por el framework y el desarrollo se base principalmente en el modelo de programación MapReduce.Hadoop ofrece las interfaces Hadoop Streaming [9] y Hadoop Pipes [10]. Hadoop Streaming es una utilidad que nos permite escribir las funciones de *map* y *reduce* en otros lenguajes diferentes a Java [41]. Hadoop Streaming usa interfaces estándar de Unix para la entrada y salida y así comunicarse con el resto de componentes de Hadoop [41]. Hadoop Pipes permite realizar procesamiento similar, pero está encaminado al uso de **C++** y la comunicación con el resto de componentes de Hadoop se realiza mediante sockets[41]. En el presente proyecto se utiliza lenguaje de programación puramente funcional como Haskell para mantener un código que sea entendible y mantenible en el tiempo, con muy buen rendimiento. Esto a pesar de no usar lenguajes que son la elección tradicional cuando se necesita rendimiento.

Hadoop cuenta con su propio sistema de archivos distribuidos [41], para facilitar el acceso a todos los nodos que ejecuten las tareas de *map* y *reduce* [41]. El sistema de archivos distribuido provee encripción de punto a punto, ofreciendo seguridad para los datos que contienen información acerca del ADN de los pacientes.

[9] Sitio de Hadoop Streaming: https://wiki.apache.org/hadoop/HadoopStreaming

[10] Sitio de Hadoop Streaming: https://hadoop.apache.org/docs/r1.2.1/api/org/apache/hadoop/mapred/pipes/package-summary.html

2.3.3 USO DEL LENGUAJE FUNCIONAL HASKELL

El uso de un lenguaje funcional aportará ventajas inherentes a sus características. Haskell es un lenguaje puramente funcional [17], lo cual significa que cualquier acción que tenga efectos secundarios, como leer o escribir de un disco, están bien delimitados en un contexto específico [17]. Lo que facilita poder razonar sobre las partes que generalmente se convierten en un cuello de botella en la mayoría de aplicaciones [34].

Además, se cuenta con otras características que harán más fácil el desarrollo y la posterior extensión de la aplicación. Por ejemplo, un sistema de tipos que usa el algoritmo de Hindley-Milner para inferencia de tipos [2], incluso siendo estáticamente tipado. Lo que significa que los errores más comunes en programación y varios de lógica pueden ser detectados en tiempo de compilación y no en tiempo de ejecución [24], donde sería más difícil de encontrarlos. Un mejor sistema de tipos permite representar de una manera más completa el dominio del problema, esto significa que errores de lógica pueden ser encontrados en tiempo de compilación.

Dentro del lenguaje, el manejo de efectos secundarios como acciones entrada y salida (IO) o mantener estado dentro de una aplicación están bien definidos [17]. Es más fácil para el compilador poder optimizar, en tiempo de compilación, el código para paralelizar acciones que no afecten a otras [24]. El lenguaje nos provee con muchas primitivas y construcciones para poder tomar ventaja de los mismas con conceptos que vienen directo de la Teoría de Categorías [24] como son: Mónadas, Monoides, Funtores, Plegables, Aplicativos y Flechas. Estos conceptos de la matemática nos proveen con abstracciones que evitan que se repita código innecesariamente al tiempo que sus propiedades matemáticas aportan una base sólida para mostrar la correctitud de la representación del dominio del problema en el sistema de tipos [28].

Cada uno de estos conceptos es merecedor de un estudio detallado acerca de cómo se relacionan con conceptos reales de la programación y qué están representadas dentro del sistema de tipos de Haskell como podemos ver en la figura 2.2:

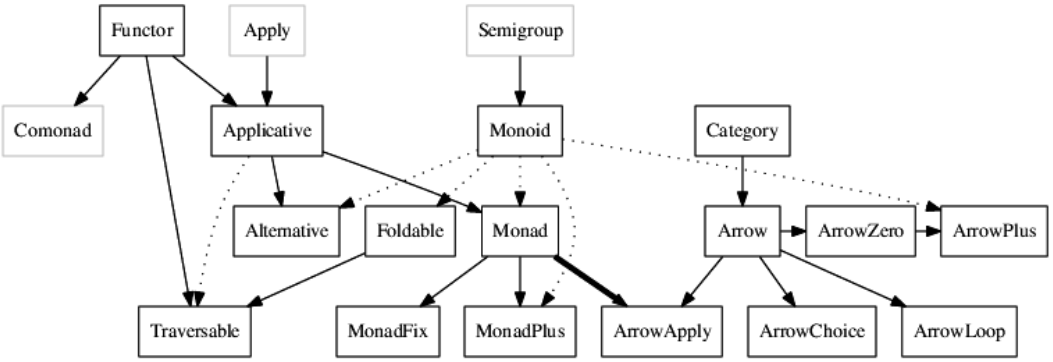

Figura 2.2: Typeclass completo de Haskell[42]

Una de las características escogidas para la aplicación es el uso de un lenguaje funcional [19]. La elección de Haskell como lenguaje para el desarrollo de la aplicación es bastante apropiado al ser un lenguaje puramente funcional con un gran ecosistema de librerías ampliamente usadas [28]. Además, aporta características que hacen que el desarrollo y extensión de la aplicación sea bastante más fácil, junto con su sistema de tipos y abstracciones para programación.

Una ventaja inherente al uso de Haskell es que está cimentado sobre una base teórica de conceptos que abarcan desde el cálculo lambda y teoría de tipos hasta la teoría de categorías [28]. Esto no significa que para que alguien pueda desarrollar en el lenguaje deba concocer a fondo estos temas, pero puede ayudar para un desarrollo de software más robusto. El sistema de tipos, el paradigma de programación y muchos conceptos que parten de las matemáticas; son características valiosas de aprender a manejar como Edsger W. Dijkstra notó en una carta dirigida a su facultad cuando se reemplazó la enseñanza de Haskell por la de Java para la carrera de Ciencias de Computación [11]. En lenguajes donde un prototipo de programa puede ser realizado de manera rápida como **Python** o **Ruby**, a largo plazo, el mantenimiento de la herramienta puede ser costoso de mantener y retrasar el desarrollo en ella [13]. Esto se puede solucionar parcialmente al realizar un desarrollo guiado por pruebas [3]. Sin embargo, con una aplicación lo suficientemente compleja el conjunto de pruebas comenzaría a realizar parcialmente las tareas del

[11] Carta de Dijkstra: https://www.cs.utexas.edu/users/EWD/OtherDocs/To%20the%20Budget%20Council%20concerning%20Haskell.pdf

compilador al validar que entradas y salidas sean del tipo que se espera [3].

Incluso conceptos como el de MapReduce son fácilmente entendibles con las abstracciones antes mencionadas como Plegables y Aplicativos [23]. Es decir que el modelo de programción MapReduce puede ser fácilmente representado dentro de Haskell. Además, el uso por defecto de una evaluación perezosa [12] en Haskell y una restricción para la aplicación de efectos en IO, fuerzan a un razonamiento más holístico al momento de desarrollar software [19]. Algo crucial si se están desarrollando aplicaciones que deben tener un gran rendimiento.

2.3.4 EFICIENCIA EN EL MANEJO DE DATOS

Dado que la mayor cantidad de trabajo es el procesamiento de la información desde los archivos que contienen las variaciones, la manera en que se trate los archivos de entrada, para su procesamiento, es crucial. Haskell cuenta con diversas librerías para el procesamiento de datos[13], algunas manejan grandes cantidades de peticiones. Un ejemplo de esta eficiencia se ve en el manejo de datos en Facebook[14], donde se puede apreciar cómo no sólo se maneja un gigantesco sistema de reglas para combatir el *spam*, también, se debe manejar una gran cantidad de datos en formato JSON (JavaScript Object Notation), y es otra librería escrita en Haskell la encargada de manejar esa gran cantidad de datos[15].

En el caso del manejo de la gran cantidad de datos que involucran los archivos con variantes de genes, se usará la librería **attoparsec**, que cuenta con un desempeño excepcional en el manejo de datos. La librería puede obtener resultados mejores que código escrito en **C**. Se toma como ejemplo una comparación para procesar

[12] La evaluación perezosa (*Lazy Evaluation*) significa que el valor de una computación no es evaluado hasta el último momento posible en que es requerido. Esto difiere de la evaluación temprana (*Eager evaluation*) que es el tipo de evaluación más común.

[13] Librerías para el procesamiento de datos: https://wiki.haskell.org/Applications_and_libraries/Compiler_tools

[14] Fighting spam with Haskell at Facebook: https://code.facebook.com/posts/745068642270222/fighting-spam-with-haskell/

[15] aeson: Fast JSON parsing and encoding https://hackage.haskell.org/package/aeson

peticiones HTTP en **C** y uno en Haskell usando **attoparsec**, donde algo más de 40 líneas de código *expresivo* en Haskell son más veloces que más de 2000 líneas de código a la medida escrito en **C**[16]. Esto se puede observar en la figura 2.3. Sin embargo, hay que tomar estos resultados con cierto cuidado pues siguen siendo pruebas artificiales. En el caso de *http-parser naive* se tiene que el parseador de peticiones asigna memoria para la petición y sus cabeceras, la cual es liberada una vez que la petición ha sido totalmente parseada. Mientras tanto, *http-parser null* representa una serie de llamadas vacías, lo que significa que es el mejor rendimiento que se puede obtener.

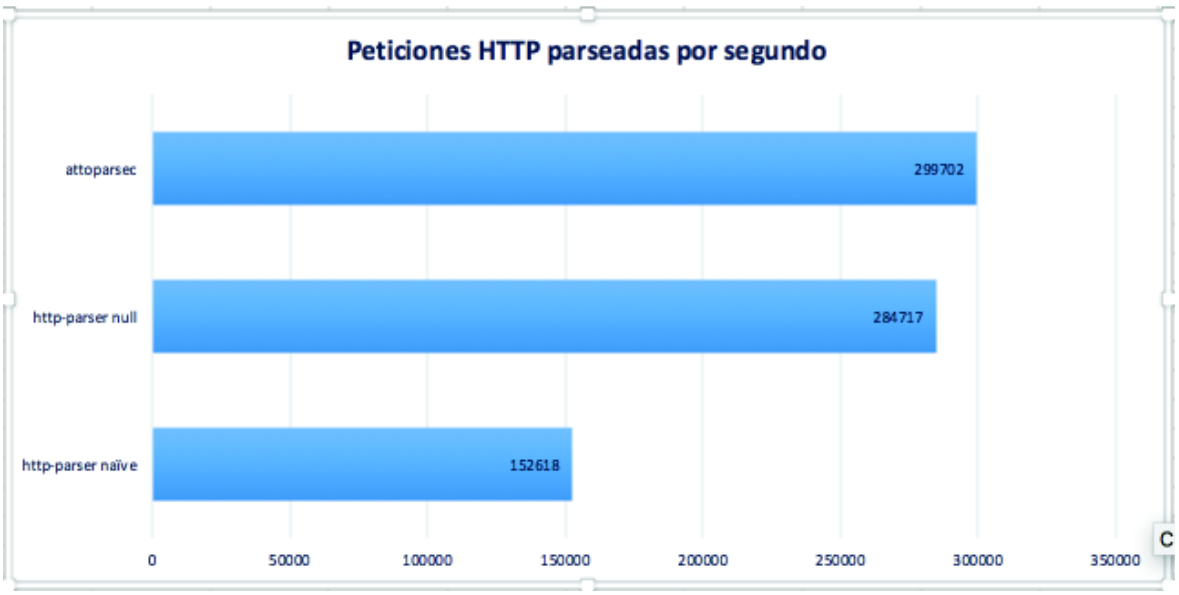

Figura 2.3: Peticiones parseadas por segundo (más es mejor)

El procesamiento de los datos se realiza usando el concepto de *Parser Combinators*. Un *parser* de *cosas* es una función que toma una cadena de texto y retorna una lista de pares con *cosas* y cadenas de texto no consumidas [28]. Un *Parser Combinator* es un *parser* resultante de la combinación de dos o más *parsers* [24]. Si un *parser* puede retornar cualquier *cosa*, se dice que es polimórfico para **a**, siendo **a** cualquier tipo de datos [24]. En Haskell se puede escribir el tipo de *Parser* como:

[16]Comparación entre procesadores HTTP en Haskell y **C** http://www.serpentine.com/blog/2014/05/31/attoparsec/

```
type Parser a = String -> [(a, String)]
```

El desarrollo de aplicaciones que procesen estas cadenas de texto como es el archivo VCF es beneficiado además por la abstracción de las mónadas directamente de la teoría de categorías [24]. Esto da lugar al concepto de procesadores monádicos combinatorios (*Monadic Parser Combinators*) [28]. Esto hace que escribir aplicaciones para el procesamiento de archivos de texto, como los archivos VCF, sea más sencillo que escribir una expresión regular. Adicionalmente, estos procesadores combinatorios pueden procesar gramáticas más complejas que las que son posibles de procesar con expresiones regulares [24], por ejemplo un JSON.

2.4 PLAN DE DESARROLLO

Una vez levantados los requisitos funcionales y no funcionales de la aplicación en la sección 2.1 se procede a escribir las historias de usuario que serán usadas durante el desarrollo.

Se utiliza el marco de trabajo SCRUM, con varias herramientas que son especialmente útiles para proyectos de gran tamaño con un tiempo de desarrollo mucho más largo que el estimado en el cronograma, de las que sólo se harán uso las partes más relevantes para el desarrollo de la aplicación planeada. El desarrollo en SCRUM se realiza en iteraciones o *sprints* que sirven como marcas de tiempo para revisar el avance en el proyecto [33].

Además de iteraciones, SCRUM consta de roles, eventos y artefactos [33]. Debido a que el proyecto se desarrolla por una sola persona, no es necesario el uso de roles. No obstante, SCRUM consta de algunas reuniones o eventos de planeación, así como, de artefactos que ayudan a llevar una idea del estado del proyecto como guías del mismo:

- Artefactos:
  - Producto Backlog
  - Sprint Backlog
  - Burndown Chart
- Reuniones:
  - Planeación del Sprint (Sprint Planning)
  - Scrum Diario (Daily Scrum)
  - Retrospectiva del Sprint (Sprint Retrospective)

2.4.1 PRODUCT BACKLOG

Es una lista de historias de usuario que describen todas las tareas que son necesarias llevar a cabo en el desarrollo del proyecto. Las historias de usuario contienen la información necesaria acerca de lo que se debe hacer, así como sus dificultad, prioridad, propósito y tipo [33].

Las historias de usuario pueden tener diversos tipos, como “feature”, “bug”, “chore” [33]. Cada uno de estos corresponde a diferentes necesidades del proyecto. En el caso del presente proyecto, el primer paso será un “spike” que proviene de XP (Extreme Programming) [37]. Este “spike” ayudará a tener un panorama claro de como funcionan los archvios VCF. Se prosigue con el desarrollo de una librería para el procesamiento de archivos VCF. Esta será separada de la aplicación principal, para que sea reutilizable posteriormente. Una vez que la librería para el procesamiento de los archivos termine, se puede proceder con la implementación del “mapper” y “reducer”.

Es necesario resaltar que el Product Backlog no tiene las historias en ningún orden específico. Además, las historias tampoco están divididas por iteración. Para llevar el registro de dichas historias se va a hacer uso de **Pivotal Tracker**[17]. Se escoge dicha herramienta por la facilidad que tiene para representar lo que se necesita de SCRUM en el Proyecto. También, se la escoge por no ser tan complicada de usar como otras herramientas disponibles como Jira.

[17] Sitio web: https://www.pivotaltracker.com/

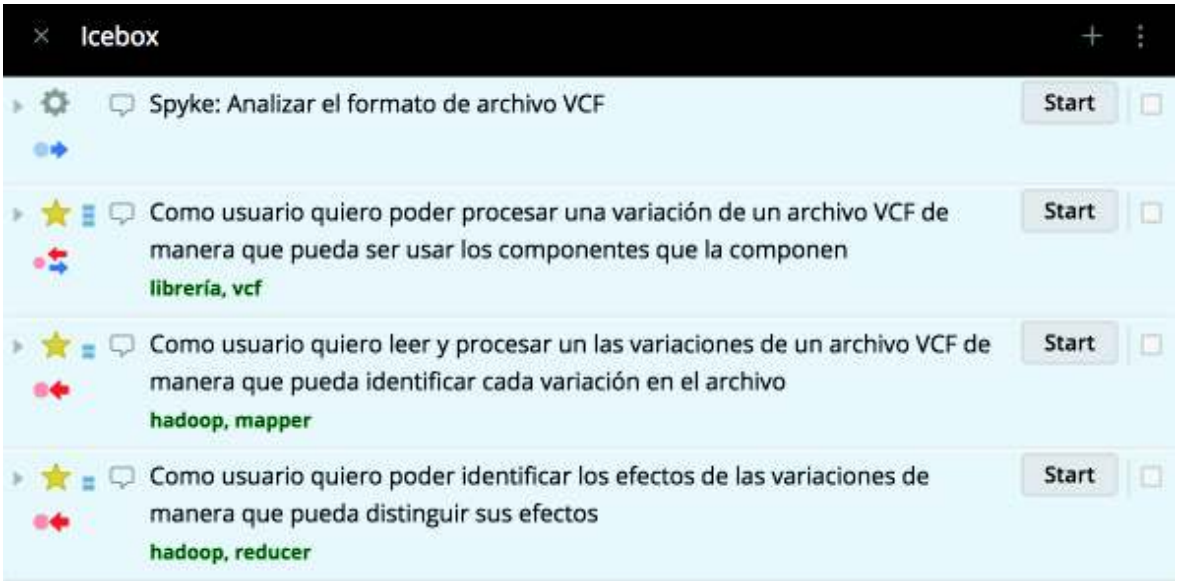

Figura 2.4: Product Backlog de la aplicación en Pivotal Tracker

2.4.2 SPRINT BACKLOG

A diferencia del *Product Backlog*, el *Sprint Backlog* cambia por iteración o sprint [33]. Es decir, existirá un Sprint Backlog para cada iteración. En caso de que existan historias que no pudieron ser completadas en una iteración, dichas historias son movidas a la siguiente iteración [33]. El porqué de no completar la historia será analizado en el Sprint Retrospective para solucionar aquello que impidió su realización [33].

Debido a que son pocas historias de usuario, se define solamente 2 iteraciones. Cada iteración deberá durar alrededor de dos semanas. Para el orden en el que se deben elegir las historias, se toma en cuenta aquellas tareas que aportan más valor al proyecto. Así también, se eligen aquellas tareas que deben ser terminadas para que otras puedan ser comenzadas [33].

Finalmente, se obtiene una lista de tareas ordenada por iteraciones y en el orden en el que deben ser realizadas.

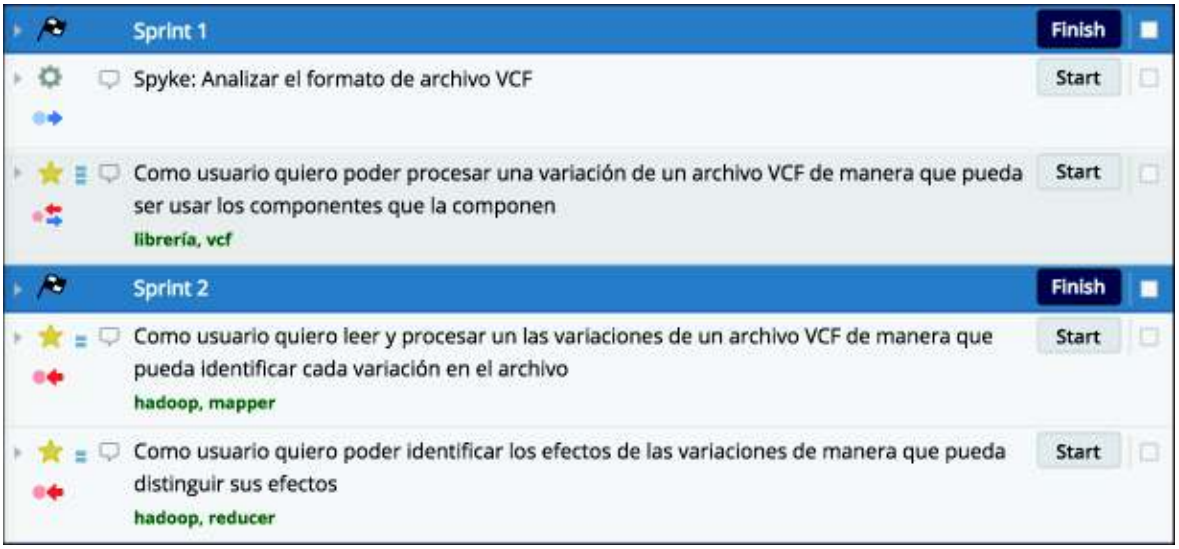

Figura 2.5: Sprint Backlog para Sprint 1 y 2 de la aplicación

Una descripcion más detallada de las historias de usuario se encuentra disponible en los Anexos .2.

2.4.3 REUNIONES

Como se estabeció anteriormente, el proyecto se desarrolla por una sóla persona. Esto significó la eliminación del uso de roles de SCRUM. Además de los roles, esto también siginifica que muchas de las reuniones que son parte de SCRUM, no será posible realizarlas.

Las reuniones diarias no caben bajo dicha premisa, puesto que estas se usan para sincronizarse con el equipo respecto al trabajo hecho, al que se hará y acerca de bloqueos. La reunión de planeación de las iteraciones será reemplazada con una la prioritización y agrupación de historias por iteración. Finalmente, se tiene a las reuniones de retrospectivas que sirven para analizar con el equipo aquello que fue mal, lo que fue bien y lo que se puede mejorar. Sin un equipo, esto quedará plasmado en el siguiente capítulo, en las dificultades que se encontraron durante el desarrollo.

2.5 ALGORITMO DE BÚSQUEDA DE VARIACIONES

La aplicación a desarrollar debe buscar aquellas variaciones en el ADN que estén asociadas con determinadas patologías y debe estar diseñada para ser dividida en las dos tareas que realiza una aplicación de MapReduce.

2.5.1 CONSIDERACIONES

- La asociación de variaciones a ciertas patologías puede ser realizada por servicios externos. Ensembl provee un API para consultar el efecto de una variación[18]. Sin embargo, la API no permite hacer una consulta por lote de variaciones, por lo que cada variación deberá realizar una petición independiente al API. Con el número de variaciones que se tienen dentro de un archivo de más de 100GB, este método no es viable.

  Emsembl también ofrece varias formas más de identificar las variables, como un archivo VCF donde se especifica las variaciones con efectos clínicos, pero que no especifica el efecto en sí[19].

- El análisis de la información debe ser pensado como un flujo de datos y manejado como tal dentro del **mapper** y el **reducer**. Una manera de visualizar este procesamiento de datos como un flujo de datos, es similar a los *pipes* de UNIX.

- La aplicación debe estar dividida en un **mapper** y **reducer**. Una aplicación puede constar de múltiples pasos cada uno con un mapper y un reducer, es decir, que nuestra aplicación puede ser resuelta en un paso o múltiples pasos.

  Por lo tanto, el algoritmo estará dividido por línea leída del STDIN, tanto para el mapper como para el reducer.

[18] Emsembl VEP API: https://rest.ensembl.org/#VEP

[19] Bases de datos de variaciones en Emsembl: http://www.ensembl.org/info/genome/variation/sources_documentation.html#homo_sapiens

- El procesamiento del archivo se realizará en una librería que se desarrollará para acompañar la aplicación.

2.5.2 ALGORITMO

Mapper

Data: Línea de archivo VCF por STDIN.

Result: Clave-valor en el STDOUT donde el id de la variación es la clave y la variación completa es el valor.

Inicio;

$resultadoDeProcesamiento \leftarrow Intentarparsearlalnea$;

if *resultadoDeProcesamiento es exitoso* **then**

  $variacion \leftarrow extraervariacindelprocesamientoexitoso$;

  $id \leftarrow extraeriddevariacin$;

  Escribir (id, $variacion$) al $STDOUT$;

else

  Escribir una línea vacía al $STDOUT$;

end

Algorithm 1: Algoritmo del mapper

Reducer

Data: Línea con llave valor, siendo la llave el identificador de la variación y el valor la variación.

Result: Efecto de la variación si esta tiene efecto relacionado con la salud.

Inicio;

$resultadoapi \leftarrow LlamaalAPIdeensembl!$;

if *resutadoAPI no tuvo errores* **then**

  $efectoDeLaVariacion \leftarrow extraerelefectodelresultadodelAPI$;

  if *efectoDeLaVariacion se expresa en la salud* **then**

    Escribir al $STDOUT$ la información de la variación $parseada$ con su efecto;

  else

    Escribir al $STDOUT$ una línea vacía;

  end

else

  Escribir al $STDERR$ el id del gen que falló;

end

Algorithm 2: Algoritmo del reducer

CAPÍTULO 3. IMPLEMENTACIÓN

3.1 DEFINICIÓN DE AMBIENTES DE EJECUCIÓN Y DESARROLLO

Para la definición de los ambientes de ejecución, se debe considerar algunos requerimientos de la aplicación definidos previamente en la seccion 2.1. Se especifica que la ejecución de la aplicación en entornos UNIX, en los cuales se tiene STDIN y SDTOUT (2.3.2). Como ejemplos de sistemas UNIX, en la actualidad, tenemos a sistemas operativos como GNU/Linux, Solaris, OS X, etc. Por lo que la necesidad de ser ejecutado en un ambiente UNIX elimina a un sistema operativo de escritorio popular como **Microsoft Windows**, pero deja la posibilidad del uso en muchos entornos que son UNIX.

Tanto **Haskell** como **Hadoop** se encuentran disponibles para Microsoft Windows. Sin embargo, lo que se va a desarrollar es una aplicación que use Hadoop Streaming. Hadoop Streaming usa por defecto STDIN y STDOUT como interfaces de comunicación entre el proceso de map, shuffle y reduce. Usualmente, la comunicación entre estos pasos se realiza a través del código Java ejecutándose sobre la máquina virtual de Java.

3.1.1 HASKELL

El lenguaje seleccionado para la implementación de la aplicación es Haskell. Para comenzar es necesario descargar las herramientas que nos permitan desarrollar

en este lenguaje. El sitio web de **Haskell** nos detalla [17] las diferentes plataformas para las que está disponible, y las diferentes opciones que hay para su instalación. Nos ofrece 3 opciones para poder instalar el entorno de desarrollo de Haskell [17]:

- **Instaladores mínimos**: Es posible instalar solamente el compilador por defecto de Haskell (**GHC**) y **Cabal**, un instalador de paquetes y la herramienta de construcción para software en Haskell. Para la instalación individual se puede utilizar el gestor de paquetes por defecto del sistema operativo que se esté usando.
- **Stack**: Esta herramienta usa una selección curada de versiones de paquetes que funcionan correctamente juntos. Esto hace más fácil evitar dependencias desactualizadas o librerías que rompen compatiblidad con nuevas versiones. El ejecutable de **stack** se puede instalar con el gestor de paquetes de sistema. Stack está construido sobre GHC y Cabal, además, está pensado para toda clase de proyectos.
- **Plataforma de Haskell**: Esta opción, instala el compilador **GHC** junto con cabal y algunas otras herramientas para el desarrollo, así como librerías iniciales en una localización global para todo el sistema. Esta era la opción más común antes de la aparición de Stack.

Todas las opciones señaladas anteriormente están disponibles para: Microsoft Windows, GNU/Linux, OS X. Las instrucciones de descarga e instalación se encuentran en su página de descargas [1].

3.1.2 HADOOP

Para el caso de **Hadoop**, el sitio web nos dirige hacia descargas del código fuente [12]. Al ser una herramienta desarrollada en Java [41], los archivos dentro de los distribuibles de **Hadoop** solo tienen como requisito contar con el entorno de

[1] Sitio de descargas de Haskell: https://www.haskell.org/downloads

desarrollo de Java (JDK) instalado en el ambiente donde se vaya a usar Hadoop. Java provee la máquina virtual necesaria para ejecutar las aplicaciones de Hadoop, además de las librerías para poder desarrollar los “trabajos” que se mandan para su procesamiento en Hadoop. Un “trabajo” o “job” se refiere a una tarea de procesamiento que será ejecutada en Hadoop. Java está disponible para las mismas plataformas que Haskell: Microsoft Windows, GNU/Linux y OS X.

Las instrucciones para la instalación se encuentran en su documentación[2]. Hadoop, puede ser instalado como nodo único y de una manera multinodo. Para realizar las pruebas aplicación locales, tan solo es necesario tener instalado a Hadoop como node único.

3.1.3 ENTORNO DE EJECUCIÓN Y DESARROLLO

El sistema se desarrollará en un sistema operativo Mac OS X. Mac OS X es un sistema UNIX que permite la instalación de todos los componentes necesarios para poder desarrollar la aplicación. Se elige a Mac OS X como sistema de desarrollo debido a que las pruebas de rendimiento serán realizadas en sistemas basados en GNU/Linux. De esta manera, se puede comprobar que la aplicación desarrollada es portable entre sistemas “UNIX”, dado que esa es la única restriccione que impone Hadoop Streaming.

Instalación del entorno

Para realizar la instalación en el entorno de desarrollo seleccionado, se puede usar un gestor de paquetes para el sistema (todo este procedimiento está descrito en la página de descargas de Haskell[3]). El entorno de desarrollo seleccionado es Mac OS X, que tiene el gestor de paquete “brew”. Una vez instalado “stack” es necesario actualizar las referencias a paquetes que tiene y finalmente hacer una configuración inicial de la herramienta. Para eso es necesario usar los siguientes

[2]Descarga e instalación de Hadoop: http://doctuts.readthedocs.io/en/latest/hadoop.html
[3]Sitio de descargas de Haskell: https://www.haskell.org/downloads

comandos

```
$ brew install stack-haskell
$ stack upgrade
$ stack setup
```

Una vez instalado “stack” para el desarrollo en haskell, se puede comprobar que las herramientas necesarias están instaladas, y que están disponibles en el sistema. Ejecutamos, desde la línea de comandos: ’*ghc –version*’ y ’*cabal –version*’, siendo el compilador y el gestor de paquetes y librerías respectivamente. El resultado nos debe decir la versión de estas herramientas que tenemos instaladas.

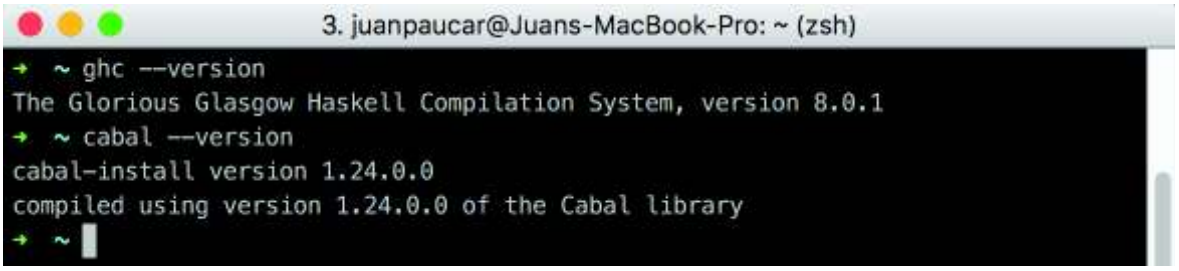

Figura 3.1: Comprobación de la instalación de herramientas en el sistema

Con el compilador y el gestor de paquetes disponibles en el sistema, se tiene las herramientas para el desarrollo y la construcción tanto de librerías, así como de ejecutables independientes.

El gestor de paquetes para Haskell, **cabal**, permite agregar librerías locales de modo que no hace falta publicar la librería en el repositorio general de Haskell. Lo que se especifica para Hadoop Streaming es un par de ejecutables para los pasos de **map reduce** [41]. Estos ejecutables podemos desarrollarlos por separado de la librería y compilarlos para pasarlos a Hadoop Streaming.

Para instalar Hadoop en el Mac OS X se puede usar de igual manera el gestor de paquetes **brew**. Cabe recordar que este comando instalará siempre la última versión disponible que se tiene. La última versión disponible al momento de ejecutar pruebas con la aplicación, es la versión 3 de Hadoop. En el caso de las versiones

disponibles en servicios enla nube, todavía es la versión 2.8[4]. En cualquier caso, no es mayor problema dado que la mayor cantidad de cambios afectan al manejo del sistema de archivos distribuido.

```
$ brew install hadoop
```

3.2 IMPLEMENTACIÓN DE LAS HISTORIAS DE USUARIO

3.2.1 PARSER COMBINATORS

Un *Parser Combinator* es una función de orden superior que acepta varios *parsers* como argumentos de entrada, y retorna un nuevo parser compuesto como resultado de la función [28]. Estos *parsers* individuales son funciones que toman una cadena de texto y retornan una estructura con sentido dentro del lenguaje de programaición. A su vez, estos *Parser Combinators* pueden ser compuesto para generar parsers más complejos [28]. Esta combinación de *parsers* permite crear *parsers* recursivos descendientes que pueden procesar estructuras con anidación o recursivas. Un ejemplo de estas estructuras son los JSONs que son estructuras recursivas. Sin embargo, los archivos VCF no tienen estructuras recursivas, pero nos tener múltiples *parsers* nos permite descomponer el procesamiento de cada parte de una línea en un archivo VCF y probar la que la implementación del parser es correcta por cada parte la línea, algo que no es posible con una expresión regular [28]. Esto también permite en el futuro refactorar el código en caso de regresión o de que la especificación cambie [3]. Además pruebas sirven de documentación de las funciones contenidas en la librería [3].

Por lo anteriormente expuesto, se decidió tomar librerías que usen "Parser Combinators". En un lenguaje de programación funcional es natural contar con funciones de orden superior [11]. Estas funciones de orden superior son aquellas que toman uno o más argumentos y retornan una función. En lenguajes donde las funciones

[4]EMR en Amazon Web Services: https://aws.amazon.com/es/emr/details/hadoop/

son elementos de primer orden es sencillo entender este concepto [11]. Sin embargo, en otros lenguajes como Java donde no se tiene esta noción, implementar un comportamiento similar requiere el uso de grandes librerías bastante especializadas que provean las bases para construir la noción de una función como elemento de primer orden.

En Java existe una librería bastante madura para el procesamiento de datos llamada "JParsec"[5]. La librería hace un uso intenso de genéricos en Java. Sin embargo, la librería presenta algunos incovenientes como el no ser adecuada para procesar grandes archivos[6]. Al no tener un polimorfimo paramétrico como el de Haskell, en esta librería se hace un uso extensivo de Clases y Genéricos. Esto no tiene necesariamente un costo extra en tiempo de ejecución, debido a que al momento de compilación, el compilador de Java aplica un **borrado de tipos** o *type erasure*. Sin embargo, por el diseño del lenguaje y el estar hecho para la programación orientada a objetos, significa que habrá mucho más código al momento de escribir un parser y no serán realmente funciones sino una composición recursiva de objetos.

Es por esto que se procede a dividir el desarrollo del proyecto en la implementación de una librería que pueda procesar los archivos. La librería, tras procesar una entrada en el archivo VCF, debe ser capaz de retornar una estructura a la que se pueda procesar en otras tareas durante el procesamiento de estos archivos en Hadoop.

De esta manera, los archivos que requiere Hadoop que para el mapper y reducer, simplemente deben hacer uso de esta librería para pasar de una línea de texto en el archivo a una estructura con la cual trabajar.

Dentro de las librerías que se puede usar en Haskell para el procesamiento de datos que usan Parser Combinators. Queda a libre elección si se prefiere contar con mensajes de errores mas comprensibles o mayor velocidad de procesamiento.

Mensajes de errores más fáciles de comprender, significa que será más fácil poder

[5]Repositorio de JParsec: https://github.com/jparsec/jparsec

[6]Ticket para que el consumo de archivos sea incremental: https://github.com/jparsec/jparsec/issues/4

corregir un error dentro del archivo que se procesa, al tener más claro el punto específico y la razón por la que falla. Sin embargo, en archivos tan grandes es poco práctico llevar estas correciones de una manera manual; fácilmente, podemos descartar pocas entradas con errores en un archivo de millones de entradas.

Entonces, se toma la decisión de usar una librería con mayor rendimiento a cambio de mensajes de errores menos claros y se escoge Attoparsec. Se puede obviar los mensajes de errores porque cada parte del parser será probada con diferentes entradas para tener certeza de que funciona como se espera. Attoparsec es una de las librerías que se usan con frecuencia en Facebook para el procesamiento de Spam. Además de Attoparsec, Haxl, librería también hecha por Facebook, aprovecha la base matemática de ciertas abstracciones en Haskell para entregar gran rendimiento en procesamiento paralelo y concurrente de una manera simple[7] y podría ser una alternativa para trabajos futuros que deseen implementar su propia estrategia de paralelismo.

3.2.2 DESARROLLO DE LA LIBRERÍA

Para comenzar con el desarrollo de un nuevo proyecto (en Haskell) se puede usar el gestor de paquetes **cabal**. Primero se debe crear un directorio nuevo y luego ejecutar:

```
$ cabal init
```

Esto nos presentará una serie de opciones que debemos llenar para crear la librería que nos ocupa. Entre las opciones que presenta están el nombre del paquete, la versión inicial de la librería, la licencia, el nombre del autor, el correo del encargado de mantener el paquete, un URL del sitio web del paquete, la sinopsis que describe al paquete, la categoría del paquete, el tipo de paquete (librería o ejecutable),

[7] Cómo se usa Haxl en Facebook: https://code.facebook.com/posts/745068642270222/fighting-spam-with-haskell/

directorio base del código, versión del lenguaje de Haskell (98 o 2010)[8], y campos adicionales si se necesitasen.

Debido a que el sistema de tipos de Haskell es bastante diferente a lo que se tiene en lenguajes orientados a objetos, podemos comenzar simplemente definiendo el sistema de tipos de la aplicación. A diferencia de un sistema de tipos en lenguajes como **C**, en **Haskell**, el sistema de tipos puede representar los objetos en el dominio del problema que se planea resolver [24] y no sirven simplemente como estructuras para el manejo de memoria.

El sistema de tipos de Haskell nos permite evitar muchos errores, comunes en otros lenguajes, desde el momento de compilación [24]. El sistema de tipos cuenta con otras muchas características como:

- **Tipado fuerte:** Ofrece garantías para atrapar errores en el momento de la compilación para no escribir un código que haga conversiones entre tipos de manera inapropiada. Por ejemplo, usar enteros como una función o intentar realizar una coerción de tipos implícita como en C o Java es común en caso de que, por ejemplo, se espere un *int* y se entregue un *float* [28].
- **Tipos estáticos:** Esto significa que el compilador conoce el tipo de cada expresión en tiempo de compilación y antes de la ejecución de cualquier código. El compilador detectará estos errores cuando se intente usar expresiones que no coincidan en los tipos [28].
- **Inferencia de tipos:** El compilador de Haskell, además, puede inferir automáticamente los tipos de casi todas las expresiones en un programa [28]. En la inferencia de los tipos usa el algoritmo de Hindley-Milner para sistemas de tipos basados en el cálculo lambda y que soporten polimorfismo [2].

Por estas razones, la primera parte del desarrollo de la librería involucra la especificación de los tipos que representarán a los datos cuando se procesen los archivos VCF. Al ser una librería podemos usarla posteriormente en la aplicación y utilizar

[8]Explicación de los cambios en Haskell 2010: https://wiki.haskell.org/Haskell_2010

los tipos base que se definen en la librería. Para comenzar con el desarrollo podemos partir de la especificación del formato VCF [16]. En la especificación se dice que el archivo VCF en un principio, está compuesto de 2 partes: la cabecera con información acerca de las anotaciones por variación y las variaciones en sí mismas. Esto podemos representarlo de la siguiente manera:

```
1 data VCF = VCF
2            { header      :: Header
3            , variations  :: [(Variation, [ByteString])]
4            } deriving Show
```

Fragmento de Código 3.1: Representación de un Archivo VCF

Hemos creado un tipo específico para la cabecera o *Header* de los archivos VCF. Como se puede observar, en la especificación de los archivos VCF [16], el *Header* contiene la metainformación acerca de los campos que se especifican en cada una de las variaciones. Las variaciones que se encuentran en el genoma del paciente son una lista ordenada con un formato similar a:

```
Y 14311734 rs201096451 T C 100 PASS AA=T;AC=17;AF=0.0137875;AN=1233;DP=4442;NS=1233;AMR_AF=0;AFR_AF=0;EUR_AF=0;SAS_AF
    =0;EAS_AF=0.0697;VT=SNP GT
```

Fragmento de Código 3.2: Entrada dentro de un archivo VCF

Cada variación tiene 8 campos obligatorios. Si el genotipo está presente, estas 8 entradas están seguidas por una adicional con información acerca del formato del genotipo. Los campos son:

- #CHROM
- POS
- ID
- REF
- ALT

- QUAL
- FILTER
- INFO

De acuerdo a las condiciones impuestas por la especificación [16] para cada campo, podemos representar a cada variación de la siguiente manera:

```
1  data Variation = Variation
2                 { chrom  :: ByteString
3                 , pos    :: Int
4                 , idx    :: [ByteString]
5                 , ref    :: ByteString
6                 , alt    :: [ByteString]
7                 , qual   :: Maybe Float
8                 , filt   :: [ByteString]
9                 , info   :: [ByteString]
10                , format :: Maybe [ByteString]
11                } deriving (Show, Eq)
```

Fragmento de Código 3.3: Representación de una variación en un archivo VCF

Aquí debemos escoger las primeras opciones para obtener un buen rendimiento. Haskell tiene varios tipos de datos para representar cadenas de texto. El tipo por defecto es *String*, pero también presenta dos opciones más para el tratamiento de datos de tipo texto como son: *ByteString* y *Text*. La primera opción, ByteString, brinda un manejo de texto eficiente y rápido, debido a se maneja el texto como una lista de bytes [28], pero el manejo de ciertas cadenas como binarios puede causar problemas de acuerdo a como se manejen las codificaciones. Por otro lado, *Text* es también una representación eficiente de texto en *Unicode* [28]. Sin embargo, para obtener el mejor rendimiento posible se opta por el uso de la librería *ByteString* debido a que la codificación de los archivos no es un problema debido a que usa UTF8 [16]. Al manejar los textos como binarios se toma trozos del texto que representan a los caracteres del mismo. Esto es similar a C, donde para el manejo de cadenas de texto se suele usar el tipo *char* que separa 1 byte para cada caracter, y no se realiza por defecto mayor procesamiento concerniente a la codificación del

texto.

Una de las facilidades que ofrece Haskell es la evaluación perezosa, lo que significa que el valor de una expresión será evaluado solo cuando se necesite. En el caso de la librería para *ByteString* tenemos opciones tanto para evaluación perezosa y para evaluación estricta. En el caso de la evaluación perezosa, es útil cuando la cantidad de memoria a ser utilizada debe ser constante [28], por ejemplo: al leer un archivo muy grande se traer el contenido por partes en vez de todo de golpe. En este caso Hadoop se encarga de la orquestación de distribución de los datos y la generación de múltliples hilos procesamiento. Es por esto que el impacto en memoria podemos configurarlo desde Hadoop para definir el tamaño de los trozos de datos en que son divididos.

En caso de que no se tuviese la opción de definir el tamaño de los trozos de datos, evaluar el contenido de un archivo de varios gigabytes de una manera estricta significaría el uso de varios gigabytes en memoria al mismo tiempo y la necesidad de configurar las máquinas que procesarán los datos de las variaciones. Es en este caso que usar la librería de *ByteString* presenta un mejor rendimiento debido a que los tamaños de caché que usa son apropiados para el caché L1 de los procesadores actuales, además de una huella de memoria constante [28]. Esto sería más apropiado en caso de que Hadoop no estuviera a cargo de la orquestación del tamaño de los datos que van a ser distribuidos entre los diferentes nodos de procesamiento.

Haskell, además, provee opciones interesantes para el procesamiento de datos como la librería de *pipes*[9] donde su principal característica es una ganrantía de un uso de recursos determinístico y un uso de memoria constante, para la orquestación de varios hilos de procesamiento Haskell. Se usará *pipes* para la lectura y escritura al STDIN y STDOUT con una huella de memoria constante. Esto es más fácil de realizar debido a la manera en cómo está construido el lenguaje, y a que las acciones en el caso de esta aplicación, Hadoop es quién se encarga de gestionar la concurrencia [41] y de exponer una interfaz en la que podemos, incluso, usar lenguajes

[9]Pipes: https://hackage.haskell.org/package/pipes

diferentes a Java cómo es el caso del presente proyecto [41].

La selección del tipo de datos *ByteString*, para el manejo de texto, permite que todo el texto pueda ser manejado de una manera más eficiente al no necesitar de una representación específica del texto dentro de los archivos. Sin embargo, a pesar de ser manejado como una cadena de bytes, podemos procesar los archivos de entrada con la librería **Attoparsec**, que es una librería para el procesamiento rápido de datos con formatos complicados [27].

Generalmente, para el procesamiento de entradas de texto, así como su posterior representación en estructuras de memoria, se haría uso de expresiones regulares. El problema de las expresiones regulares es que es difícil probar su correcto funcionamiento, así como su rendimiento. En el caso de **Attoparsec** nos permite lidiar con formatos complejos como *JSON* y representarlo en las estructuras que hayamos definido en nuestro sistema.

Es así que Attoparsec es la librería base para **Aeson**, otra librería del mismo creador de Attoparsec que se utiliza en Facebook para el procesamiento de gigantescas cantidades de información para saber si deben ser clasificadas como spam o no[10]. A la escala de datos que maneja Facebook, sacar el mayor rendimiento de los servidores es importante y el uso de estas librerías brinda una mayor seguridad acerca de la eficiencia del procesamiento de los datos.

Conceptos como Mónadas o Funtores que, a pesar de nacer desde la Teoría de Categorías [28], sirven para extender las posibilidades del lenguaje. En el caso de la librería permite que todo el análisis gramatical de los datos se haga dentro de un contexto específico llamado **Parser**. El cual se vio que era una función que toma una cadena de texto y retorna una lista de pares con “algo” procesado y el resto de la cadena por consumir. Para los archivos VCF, la especificación entrega lineamientos muy claros acerca de qué está permitido en cada campo del archivo [16]. Por ejemplo, **QUAL** de calidad menciona que en caso de que el valor sea desconocido se lo debe especificar; al saber que el valor de calidad puede o no

[10]Fighting spam with Haskell: https://code.facebook.com/posts/745068642270222/fighting-spam-with-haskell/

existir, se lo representa con un tipo $Maybe$:

```
1 qual :: Maybe Float
```

Esto puede representar la existencia o falta del valor en el campo de calidad, el cual es un valor decimal. En caso de no existir, este sería algo similar a *Just 108.3*, mientras que en caso de no existir un valor de calidad o de no poder ser convertido a un float, sería *Nothing*. Para analizar el archivo VCF en busca del valor de calidad, tendríamos algo como esto:

```
1 parseQual :: Parser (Maybe Float)
2 parseQual = (readMaybe . BS8.unpack) `fmap` takeWhile1
    isFloatNumber
```

Parser es un tipo de dato que instancia de Mónada y Funtor, por lo que se puede operar dentro de los contextos que nos ofrecen estos tipos de datos [28]. Hadoop se encarga de la complejidad de orquestación de procesamiento y distribución de datos, por lo que no es necesario que acudamos a una composición más compleja de abstracciones para tener a la aplicación funcionando. La librería nos ofrece facilidad y simplicidad a la hora de componer parsers. Como podemos ver, el resultado final es la librería para parsear los datos de los archivos VCF y poder representarlos como estructuras internas de la aplicación.

Al no usar expresiones regulares sino parsers, se puede probar que cada pequeña parte de la aplicación que se encargue de analizar cada parte de los archivos VCF. Esto se verá más adelante así como lo sencillo que es realizar pruebas que se encarguen de probar varias funciones a la vez, obteniendo pruebas de integración [3]. Esto resume cómo se hizo la librería encargada de establecer las representaciones de los datos en los archivos VCF y de analizarlos gramaticalmente para procesarlos. Se la separó de los 2 ejecutables que necesita hadoop para que ambos pudiesen usar la librería y sus definiciones.

3.2.3 DESARROLLO DEL MAPPER Y REDUCER

En el entorno base de Hadoop, lo que se desarrolla para el procesamiento de datos es la extensión de las clases *Mapper* y *Reducer,* de Java, que procesarán los datos. Para el uso de Hadoop Streaming en lugar de usar el lenguaje Java, podemos hacer uso de cualquier lenguaje mientras que los ejecutables resultantes cumplan con estas condiciones:

- Leer del STDIN o entrada estándar del sistema
- Escribir al STDOUT del sistema

El diagrama del modelo de una aplicación MapReduce (3.2) muestra los pasos que se sigue para la ejecución de una applicación que continúe con dicho modelo.

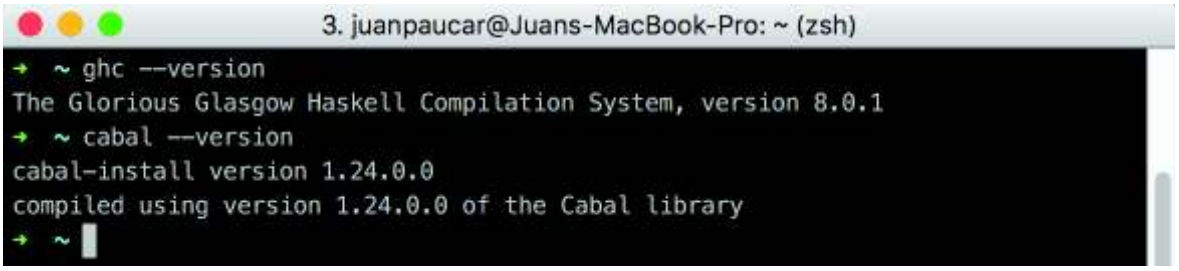

Figura 3.2: Comprobación de la instalación de herramientas en el sistema

Hadoop es una implentación del modelo MapReduce, esto significa que tendrá los siguientes pasos [23]:

1. Mapear: Se particionan los datos y se envía estas partes a los mapper que los convertirán en algo clave-valor
2. Ordenar: Se ordenan los datos particionados de acuerdo a su clave
3. Reducir: Se agrupan los datos de acuerdo a la clave y pasan a ser procesados el reducer

Hadoop se encagará de orquestar el particionamiento, así como la repartición de las tareas entre los diferentes nodos de trabajo. Posteriormente, se encargará de ordenar los resultados de la ejecución del *Mapper* por clave. Finalmente, se encargará de reducir los datos que compartan la misma clave. El paso de reducer se puede obviar mediante las configuraciones de ejecución. El resultado final de la ejecución es un archivo con las salidas del *Reducer* de los diferentes nodos [41].

Esto simplifica en gran medida el desarrollo de una aplicación que pueda beneficiarse de todas las características que vienen en Hadoop por defecto.

Debido a que la lógica para la generación de representaciones de las variaciones dentro de Haskell está en una librería separada, el desarrollo de los ejecutables para los pasos de *Map* y *Reduce* no se preocupa de la lógica necesaria para trabajar con los archivos VCF.

3.3 PRUEBAS UNITARIAS Y DE ACEPTACIÓN DEL SOFTWARE

Como se especificó anteriormente, se dividió el desarrollo de la aplicación en una librería y los scripts que la usarían. Esto significa que podemos asegurarnos de que el procesamiento de las líneas en los archivos VCF es correcto. Por otro lado, la aplicación es la implementación del algorito descrito en la sección 2.5.2. La librería está compuesta de funciones individuales que procesan diferentes partes del archivo. Cada una de estas funciones pueden ser probadas antes de ser compuestas en una función que se encargará de procesar una entrada completa en el archivo.

Por ejemplo, vamos a probar la parte de la función que se encarga de procesar la parte de las bases o nuclétidos. El ADN tiene solamente 4 bases o nucléotidos combinados dentro del genoma [6]. Esta es la referencia a la base o bases CTAG que deben estar en dicha posición. También es válido la letra N para representar que en dicha posición no debería haber una base. Según la documentación del estándar, este campo puede incluir una o a mas de una base. Además, el procesamiento debe ser insensible a mayúsculas o minúsculas.

Partiendo de esto podemos hacer la función que procese esta parte:

```
1 isBase :: Word8 -> Bool
2 isBase c = c == 65 || c == 97  || -- A or a
3            c == 67 || c == 99  || -- C or c
4            c == 71 || c == 103 || -- G or g
5            c == 84 || c == 116 || -- T or t
6            c == 78 || c == 110    -- N or n
7
8 parseRef :: Parser B.ByteString
9 parseRef = takeWhile1 isBase
```

En esta función estamos tomando letra por letra, en este caso las únicas letras permitidas son las que representan a las bases: CATG. Este parser seguirá tomando caracteres de la cadena de entrada mientras cumplan con la condición de ser una base y se detendrá cuando ya no cumpla la condición [28].

Ahora se puede hacer una prueba, para asegurarnos que está procesando de una manera adecuada y que estamos teniendo el resultado que esperamos. Para este propósito usaremos la librería **hspec**[11] que es bastante similar en la sintáxis que se requiere para escribir las pruebas a la librería rspec en Ruby donde es bastante común el desarrollo guiado por pruebas [3]. Hspec permite escribir pruebas que sirven como documentación al leerlas, es por esto que las pruebas escritas para la aplicación están escritas en el idioma Inglés, de manera que sea más natural leer la funcionalidad.

[11] Hspec: http://hackage.haskell.org/package/hspec

```
describe "parseRef" $ do
  it "should read only bases and N" $ do
    let result = parseOnly parseRef "ACTGN "
    result `shouldBe` Right "ACTGN"

  it "should be case insentitive" $ do
    let result = parseOnly parseRef "AcTgNCcAaTtGn  "
    result `shouldBe` Right "AcTgNCcAaTtGn"

  it "should fail when invalid caharacters for a base
     are passed" $ do
    let result = parseOnly parseRef "Hello"
    result `shouldBe` Left "Failed reading: takeWhile1"

  it "should parse single bases" $ do
    let result = parseOnly parseRef "A "
    result `shouldBe` Right "A"
```

Fragmento de Código 3.4: Pruebas para el parser de ref

Podemos ejecutar las pruebas de la librería con el siguiente comando:

```
stack test
```

Las pruebas fueron exitosas tal y como se evidencia tras la ejecución de las pruebas, en la figura 3.3.

```
parseRef
  should read only bases and N
  should be case insentitive
  should fail when invalid caharacters for a base are passed
  should parse single bases
```

Figura 3.3: Resultado de la ejecución de las pruebas

Los Parser Combinators, que se mencionaron anteriormente, permiten que se puedan combinar los parsers individuales en un parser compuesto que se encarga de procesar toda la línea. Por supuesto, es que el parser compuesto puede ser probado para verificar que la composición de todos los parsers funcionan como se espera que lo hagan. Además, se incluye, las pruebas de aceptación como tal, que significa poder cumplir las historias de usuario. En este caso, una característica del programa es que se pueda procesar archivos. Las pruebas de aceptación de la librería pueden ser incluídas con el resto del código. El resultado de ejecución de esas pruebas también es positivo, así que sabemos que, al menos para los escenarios planteados, la librería funciona como se espera.

```
parseVariation
  should be able to parse an entire line of variations for one patient
  should parse a variation for multiple pacients
  should be able to parse formats like those associated to diseases
  should work for tabs as separation
Multiple variations parsing
  should be able to parse multiple variations
```

Figura 3.4: Resultado de la ejecución de las pruebas

Los tests unitarios, en este caso, nos dan una base sobre la cual cerciorarnos de que las partes más pequeñas del sistema funcionan correctamente individualmente [3]. Además, nos proveen una garantía en caso de necesitar refactorizar el código que se encarga de procesar el archivo. Estas refactorizaciones pueden ser para actualizar secciones cuando el estándar sea actualizado o incluso para mejoras de rendimiento en el parser.

3.4 DESARROLLO DE LA APLICACIÓN

Una vez hecha la librería, se procede a implementar en el algoritmo definido an la sección 2.5.2. El algoritmo es realmente corto y se encuentra dividio en una aplicación para el mapper y otra para el reducer.

El mapper es el encargado de recibir los datos particionados desde el STDIN. para leer desde el STDIN se va a hacer uso de la librería **pipes**. Para o escribir basta con escribir un código muy similar a lo que uno usaría ejecutando comandos en una terminal de UNIX.

Por ejemplo, para leer del STDIN y escribir al STDOUT, sin hacer ningún procesamiento, se puede usar un código similar a este:

```
1   import Pipes
2   import qualified Pipes.Prelude as P  -- Pipes.Prelude
        also provides 'stdoutLn'
3
4   main = runEffect $ P.stdinLn >-> P.stdoutLn
```

La implementación del algoritmo, una vez que se opera por línea, es bastante similar a lo que se tiene escrito en el algoritmo. Al final, esta genera el reporte con los efectos de las variaciones.

Tras haber finalizado las historias de las dos iteraciones planeadas, se verifica si las historias cumplen con los criterios de aceptación que tenían las historias y de ser el caso, las historias son aceptadas.

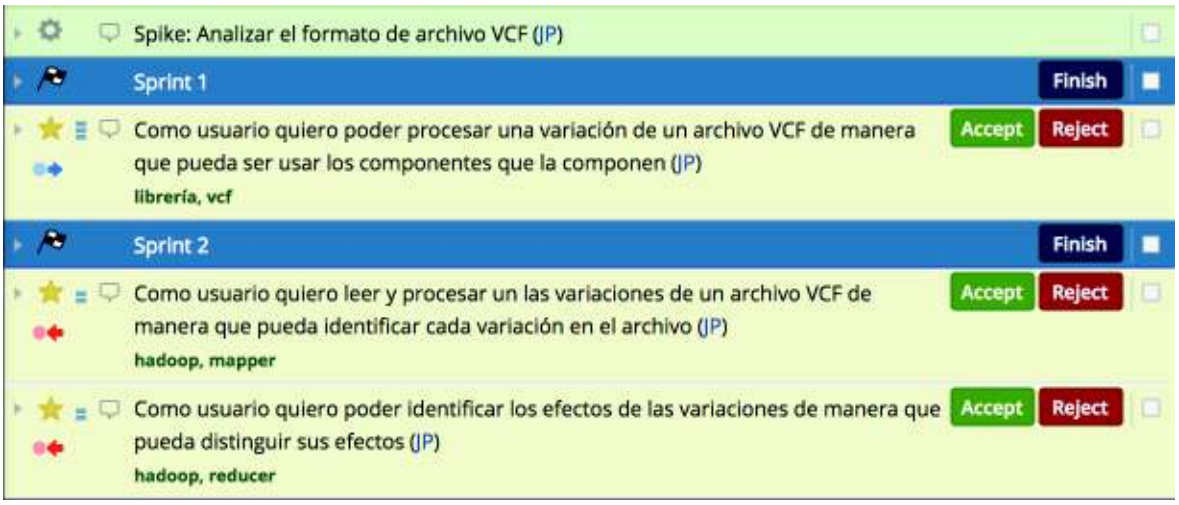

Figura 3.5: Historias de usuario finalizadas

Además, se debe verificar que los requerimientos funcionales y no funcionales hayan sido cubiertos en su totalidad. En caso de no haber cubierto alguno de los requerimientos funcionales, se vuelve a la historia de usuario que corresponde para verificar que cumplía los requerimientos funcionales. En caso de que alguno de los requerimientos no funcionales, no haya sido cubierto, se genera una nueva historia de usuario para cubrirlo. Al no ser algo que aporte valor en funcionalidad, se crea un *chore* para poder cubrir el requerimiento faltante [33].

3.5 DESCRIPCIÓN DE PROBLEMAS ENCONTRADOS DURANTE LA IMPLEMENTACIÓN

Durante la implementación de la solución se encontró con algunos incovenientes que afectaron la velocidad del desarrollo. Aquí se describirán algunos de los mismos.

- **Implementación de SCRUM**: Debido a que el equipo estaba formado por una sola persona, el uso de SCRUM no fue especialmente beneficioso. Sin embargo, se pudo haber escogido otra metodología ágil que fuese más apropiada para un equipo de trabajo pequeño como XP (Extreme Programming).

- **Tamaño de los archivos:** La cantidad de información contenido en un genoma humano es bastante grande. La información de las variaciones de un genoma humano se encuentra en el orden de los 20 GB. Esto significa un problema al momento de distribuir y procesar esta clase de archivos.

  El distribuir los archivos, puede tomar una gran cantidad del ancho de banda disponible. Esto se disminuye, en cierta manera, al comprimir el archivo antes de distribuirlo. Sin embargo, esa no es siempre una opción. Hadoop Streaming divide el archivo por partes, sin importar su cabecera. Esto significa que en la mayoría de formatos de compresión, no serán viables de usar con Hadoop Streaming. Esto se puede solucionar con ciertos formatos de compresión que

permiten ser particionados[12], aunque se debe considerar el tiempo y trabajo extra que implica el procesamiento para comprimirlos y decomprimirlos.

- **Herramientas disponibles:** Hadoop es un framework, y como tal, tiene herramientas que sirven para distintos propósitos. Así también tiene librerías internas para *mappers* y *reducers* escritos en Java y que sirven para tareas poco comunes en Hadoop. Dichas librerías así como las tareas escritas en Java pueden tener ventaja al correr sobre la misma JVM (Java Virtual Machine) que corre Hadoop. Un ejemplo es el manejo de archivos comprimidos y particionados.

  En el caso de la aplicación desarrollada, sí existen librerías para el manejo de esa clase de archivos, pero de la comunicación con la JVM. Esto dificulta algunas tareas debido a que la única forma de comunicación dentro de Hadoop Streaming es a través del STDIN y STDOUT. Es por esto que el tipo de análisis de datos debe ser pensado como un *stream* o flujo de datos que no requiera comunicación entre el procesamiento de los datos. Aunque cabe recalcar que la mayoría de análisis realizados por Hadoop son de este estilo y el hecho de requierir coordinación extra, puede ser un antipatrón en el diseño.

  Esta clase de análisis de los datos como flujo es común en Apache Spark[13], una herramienta que funciona sobre Hadoop. Spark puede ser usado con Scala, el cual es un lenguaje que combina la programación funcional con la programación orientada a objetos. Scala es lo suficientemente maduro para manejar muchos de los patrones funcionales que se encuentran en el diseño de aplicaciones que usan Haskell. Esto lo convierte en una buena opción para futuros análisis de esta clase de archivos.

[12] BZIP2 es un formato que puede ser particionado: http://www.bzip.org/
[13] Apache Spark: https://spark.apache.org/

CAPÍTULO 4. PRUEBAS DE RENDIMIENTO Y RESULTADOS

Tras la finalización del desarrollo de la aplicación en el capítulo anterior, en este capítulo se detalla las pruebas de desempeño con la aplicación. Para esto se sigue un enfoque sistemático para la medición del rendimiento del programa. El programa será ejecutado en la infraestructura de Amazon Web Services. Así, mediante el enfoque que se utilizará, se evaluará el rendimiento y comportamiento de la aplicación ante diferentes variables de manera que se tenga la capacidad de caracterizar el rendimiento para esta aplicación específica. Para finalmente obtener un modelo que prediga el tiempo de ejecución de la aplicación ante diferentes variables.

4.1 PLAN DE PRUEBAS

El plan de pruebas consiste en una serie de pasos extraídos del libro “The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling” [22]. Los pasos que se seguirán, son los siguientes:

1. Establecer Metas y Definir el Sistema
2. Listar Servicios y Salidas
3. Seleccionar Métricas de Rendimiento
4. Listar Parámetros del Sistema y la Carga

5. Seleccionar Factores de Estudio y sus Valores
6. Seleccionar las Ténicas de Evaluación
7. Seleccionar la Carga de Trabajo
8. Diseñar Experimentos
9. Analizar e Interpretar los Datos
10. Presentar los Resultados

Al término del plan de pruebas, se espera poder generar una caracterización del comportamiento de la aplicación para poder predecir el costo de ejecutar la aplicación con diferentes variables.

4.1.1 ESTABLECER METAS Y DEFINIR EL SISTEMA

El primer paso en cualquier evaluación de rendimiento es establecer las metas del estudio y definir el sistema delineando sus límites [22].

Incluso con el mismo hardware y software, la definición del sistema podría variar dependiendo de las metas que se planee estudiar. Por ejemplo, dado dos procesadores, la meta podría ser estimar su impacto en el tiempo de respuesta de usuarios interactivos. En ese caso el sistema que se estaría midiendo, consistiría en el sistema de *timesharing* y los resultados finales podrían depender de componenetes externos a los procesadores. En otro caso, donde los procesadores son idénticos excepto en su memoria L2 y la meta es decidir cual memoria L2 debe ser seleccionada, entonces el sistema sería solo el procesador y los componenetes dentro del mismo.

La elección de los límites del sistema afecta a las métricas de rendimiento tanto como las cargas usadas para comparar los sistemas. En este caso, el sistema que mediremos serán todos los nodos que corran Hadoop.

Como se estableció al principio del presente proyecto, el costo de la computación en la nube está reduciéndose [35]. Poder ejecutar en la nube singifica minimizar el tiempo de configuración de servidores e instalación de servicios. Para la ejecución del presente proyecto se utiliza el servicio EMR (Elastic Map-Reduce) de Amazon Web Services, que nos permite desplegar, de una manera bastante sencilla, toda una infraestructura de nodos corriendo Hadoop con acceso a otros servicios de alta concurrencia de Amazon, como S3 **Simple Storage Service** que nos permite guardar archivos con alta disponibilidad en la nube. Todo la ejecución e incluso el agregado de más nodos a los que se encuentren corriendo Hadoop se realizan desde una interfaz web, lo que permite que cualquier persona, necesite mínimo entrenamiento para poder poner a correr toda la infraestructura con los datos con necesite.

Las metas en el caso de nuestra aplicación serán:

1. Generar una caracterización del rendimiento de la aplicación
2. Predecir el costo de correr la aplicación con diferentes variables

4.1.2 LISTAR SERVICIOS Y SALIDAS

Cada sistema provee diferentes servicios. Por ejemplo: una computadora puede proveer el servicio de comunicación en red, así como proveer un servicio de base de datos local. Y esto significa que el procesador o cualquier recurso ejecutará un conjunto diferente de instrucciones, y el rendimiento puede ser diferente para cada servicio [22].

Entonces, el siguiente paso es listar estos servicios. Cuando un usuario hace una petición a cualquiera de estos servicos, existe un gran número de posibles resultados. Algunas de estas salidas son deseables para el análisis y otras no. Por ejemplo, un servicio de base de datos puede responder de manera adecuada a una consulta o responder de manera incorrecta debido a un estado inconsistente, incluso, puede no responder debido a deadlocks dentro de la base de datos que no le permitan

responder a la consulta de manera apropiada. Es por esto que se elige también una lista de salidas del servicio que serán requeridas.

En el caso que corresponde a presente investigación, el servicio es el proceso de Map y Reduce de Hadoop. Las salidas esperadas para el sistema ya están restringidas por la lógica de la aplicación. Donde los errores son simplemente descartados y no se lleva un log de los errores. Entonces, para la medición todas las salidas del sistema son válidas.

Estos servicios se pueden especificar desde la consola de Amazon Web Services donde se especifican los archivos mapper y reducer. En la consola se cargarán los ejecutables tras la compilación de los ejecutables en el proyecto.

El funcionamiento de EMR (Elastic MapReduce) es bastante simple y se sigue sólo tres pasos a través de la interfaz web:

1. Subir la aplicación que va a ejecutarse y los datos que va a procesar
2. Configurar el clúster con las configuraciones que se desea incluyendo: las entradas y salidas de datos, tamaño del clúster, configuraciones de seguridad, etc.
3. Finalmente, tras crear el clúster, este comenzará a ejecutar la aplicación. Mientras esta se ejecuta, se tiene acceso a métricas del progreso y del estado de los nodos del clúster. Al finalizar, se contará con los todos estos datos consolidados.

4.1.3 SELECCIONAR MÉTRICAS DE RENDIMIENTO

El siguiente paso es seleccionar el criterio para comparar el rendimiento. A estos criterios se los denomina métricas. Generalmente, las métricas suelen estar relacionadas a la velocidad, precisión y disponibilidad de los servicios. Por ejemplo, el rendimiento de una red es medido por la velocidad de transmisión, tasa de error y disponibilidad de los paquetes enviados. El rendimiento de un procesador es medi-

do por el tiempo que toma ejecutar de varias instrucciones [22].

Para la aplicación del presente proyecto, se medirá la velocidad para procesar los archivos con variaciones que se pasen al conjunto de nodos que corren Hadoop para ejecutar la aplicación.

Las métricas y datos que se recolectan al finalizar la ejecución servirán para este propósito. Dichos datos tienen una información detallada del tiempo que toma el procesamiento de los datos en las diferentes etapas que tiene una aplicación de MapReduce. Es por eso que las métricas elegidas son:

1. Tiempo de ejecución total de un bloque de variantes en el mapper
2. Tiempo de ejecución total de un bloque de variantes en el reducer

4.1.4 LISTAR PARÁMETROS DEL SISTEMA

A continuación, se debe realizar una lista de todos los parámetros que afectan al rendimiento. La lista puede estar dividia en parámetros del sistema y parámetros de la carga. Los parámetros del sistema incluyen parámetros de software y hardware, que no varían demasiado entre varias instalaciones en diferentes sistemas. Los parámetros de carga son características de los requerimientos del usuario, que varían de una instalación a otra [22].

Puede ser que algunos parámetros no hayan sido considerados en una primera versión de la lista; sin embargo, estos parámetros pueden ser agregados a la lista en una siguiente versión. A pesar de esto, se debe intentar que la lista sea tan completa como sea posible. Esto permite que tras el análisis se pueda discutir el impacto de varios de los parámetros y determinar qué datos necesitan ser recolectados antes o durante el análisis [22].

En el caso de la aplicación del presente proyecto, los siguientes parámetros pueden afectar el rendimiento de la aplicación, y en consecuencia al rendimiento de la aplicación.

- Tamaño del archivo de variaciones
- Número de servidores ejecutando la aplicación
- Tamaño de la instancias (de acuerdo al correspondiente en Amazon Web Services)
- Latencia entre los servidores que ejecutan la aplicación
- Latencia del sistema de archivos distribuido de Hadoop que es manejado por EMR
- Región de AWS en la que se encuentra cada servidor

4.1.5 SELECCIONAR FACTORES DE ESTUDIO Y SUS VALORES

La lista de parámetros se puede dividir en dos partes: aquellos parámetros que variarán durante la evaluación y aquellos que no lo harán. Los parámetros que van a variar, se llaman factores y sus valores se llaman niveles. En general, la lista de factores y sus posibles niveles es mas grande de lo que los recursos disponibles permitirían, de otra manera, la lista solo crecería hasta que fuese obvio que no existan recursos suficientes para estudiar el problema. Es mejor comenzar con una lista corta de factores y un pequeño número de niveles para cada factor, desde ahí en la siguiente fase del proyecto extender la lista si los recursos lo permiten [22].

Por ejemplo: se puede decidir tener solo dos factores el tamaño de una petición y el número de usuarios. Para cada uno de estos factores se puede escoger uno solo de dos niveles: pequeño y grande. El tamaño de la carga y el tipo de la misma podría ser fijo y no variar. Los parámetros que se esperan que tengan un alto impacto en el rendimiento, preferiblemente deberían ser seleccionados como factores [22].

Al igual que las métricas, un error común al seleccionar los factores es tomar los parámetros que son fáciles de medir y variar. Mientras que otros factores con más influencia en el rendimiento son ignorados simplemente debido a la dificultad que involucra poder medirlos. Al seleccionar factores, es importante considerar las restricciones existentes. Así también, es importante considerar el tiempo que se tiene

para el análisis. Esto incrementa la posibilidad de encontrar una solución que es aceptable e implementable.

Para el presente proyecto se tomará un camino similar al ejemplo. La carga del sistema será constante y no variará durante ningún experimento. Los factores que variarán serán el número de nodos que corren Hadoop y el tamaño de la instancia. En el caso del número de nodos se tendrá desde 1 hasta 4 nodos; el tamaño de los servidores solo tendrá dos niveles que serán dos tamaños diferentes en las instancias de Amazon Web Services la m1-medium y la m4-large. Se escoge no variar el tamaño de la carga y manetenerlo fijo, debido a que, por la arquitectura de Hadoop Streaming, aumentar o disminuir la carga, afectará al sistema de una manera líneal, ya que el uso de STDIN y STDOUT no se ven afectados por el throughput que se le da al sistema. Una carga más grande solo incrementaría el tiempo proporcionalmente y decrecer el tamaño de la carga, decrecería el tiempo.

La carga designada será un archivo con una mezcla de diferentes genomas unidos y tomados del archivo libre de genomas humanos del Proyecto Ensembl. La carga cuenta un tamaño de aproximadamente 120 GB. De esta manera, el tamaño de la carga no es trivial ni es demasiado grande para hacer que el tiempo resultante entre las mediciones sea haga demasiado largo.

Los parámetros que serán medidos y que tienen mayor influencia en los resultados de la evaluación de rendimiento junto a sus respectivos niveles se encuentran detallados la siguiente tabla:

Parámetro	**Descripción**	**Niveles**
W	Tamaño de entrada	120 GB
p	Número de servidores	1, 2, 3, 4
T	Tamaño de los servidores	m1-medium, m4-large

Tabla 4.1: Lista de parámetros y niveles

4.1.6 SELECCIONAR LAS TÉCNICAS DE EVALUACIÓN

Existen tres técnicas comúnes para la evaluación de rendimiento. La selección de la técnica adecuada depende del tiempo y los recursos disponibles para la resolución del problema, así como del nivel de precisión que se desea en la evaluación [22]. Las técnicas más comúnes son:

- Modelado analítico
- Simulación
- Medición de un sistema real

La consideración clave para decidir la técnica de evaluación debe ser en qué estado del ciclo de vida se encuentra el sistema. Las mediciones son posibles solamente si existe algo similar al sistema propuesto, por ejemplo: cuando se hace una versión mejorada del producto. En caso de que sea un nuevo concepto, un modelo analítico o simulación, son las únicas ténicas de las que se puede escoger. Esto debido a que el modelado analítico y la simulación pueden ser usados donde la medición no es posible [22].

En el presente proyecto se hará la medición de un sistema real. Se realizará la medición del sistema a desarrollado, pues en el ciclo de vida del sistema, este ya se encuentra realizado y en ejecución, por lo que no hay necesidad de realizar una simulación.

Como se mencionó anteriormente, para la medición se utilizarán los datos que se recoletan al finalizar la ejecución de la aplicación, estos datos serán luego procesados para ser organizados de acuerdo a los parámetros y cargas que se han seleccionado.

4.1.7 SELECCIONAR LA CARGA DE TRABAJO

La carga de trabajo consiste en una lista de peticiones de servicios al sistema. Por ejemplo, la carga de trabajo para comparar diferentes motores de bases de datos puede ser un conjunto de “consultas”. La carga de trabajo puede ser expresada de diferentes maneras depenediendo de la técnica de evaluación [22].

Por ejemplo, para un modelado analítico, la carga de trabajo se expresa como la probabilidad de varias peticiones al sistema. Para una simulación, se puede usar una muestra de peticiones medidas en un sistema real. En el caso de la medición, la carga de trabajo podría consistir de scripts de usuarios a ser ejecutados en los sistemas. En todos los casos, es escencial que la carga de trabajo sea representativa respecto al uso del sistema real. Para producir cargas de trabajo representativas, se necesita medir y caracterizar la carga de trabajo de sistemas existentes [22].

La carga de trabajo, en este caso, será la aplicación desarrollada en el proyecto de tesis, la misma que será subida a EMR para que la ejecute sobre el clúster de nodos de Hadoop.

4.1.8 DISEÑAR EXPERIMENTOS

Una vez que la lista de factores y diferentes niveles que estos tendrán, se debe decidir una secuencia de experimentos que ofrezcan la mayor cantidad de información con la menor cantidad de esfuerzo. En la práctica, es bastante útil conducir el experimento en dos fases. En la primera fase, el número de factores puede ser grande pero el número de niveles es pequeño. La meta es determinar el efecto relativo de varios factores [22]. En el presente proyecto se definen 3 factores del sistema que tienen especial importancia en el desempeño de la aplicación:

- Tamaño del archivo de variaciones
- Número de servidores ejecutando la aplicación

- El tamaño de la instancias (de acuerdo a la denominación en Amazon Web Services)

Gracias al uso de Hadoop no es necesario hacer ningún cambio a la aplicación para poder probar estos diferentes factores. Así también, el uso del servicio de EMR en la infraestructura de Amazon, hace que sea bastante trivial variar el nivel de cualquiera de los factores.

Experimento propuesto

La aplicación se ejecutará con cada uno de los factores y niveles descritos, dentro del servicio de EMR (Elastic Map Reduce) en la plataforma de Amazon Web Services. Hadoop ejecutará la aplicación desarrollada para identifcar las variaciones que tienen efectos sobre la salud. El experimento realizará 50 mediciones con cada nivel y cada factor. El experimento debe ser repetido varias veces para eleiminar sesgos estadísticos en el tiempo de ejecución de la aplicación. El tiempo que toma el experimento se puede expresar coo una función de los niveles de cada factor.

Siendo:

- W el tamaño del archivo de variaciones
- p el número de instancias ejecutando la aplicación simultáneamente.
- T el tamaño de la instancia.

$$t = f(W, p, T) \tag{4.1}$$

Sin embargo, el tamaño del archivo de variaciones no será modificado en ninguna ejecución. Entonces, el factor W pasa a ser una constante. Por lo tanto, la ecuación anterior puede ser reescrita en función solamente de p y T.

$$t = f(p, T) \tag{4.2}$$

Como se explica en la definición de MapReduce [23], este modelo de programación consta 3 fases principales para el procesamiento de datos. Estos son: *map*, *shuffle* y *reduce*. Esto significa que el tiempo de ejecución puede tambien ser descrito como la suma del tiempo de estas tres fases.

$$t_{total} = t_{map} + t_{shuffle} + t_{reduce}$$

Por lo tanto, se puede decir que el tiempo está en función de los factores con sus niveles y de las fases de MapReduce. Esto significa que los datos que se recolecten de los experimentos deberán incluir distinciones de estas distintas variaciones.

4.1.9 ANALIZAR E INTERPRETAR LOS DATOS

Se debe tener en cuenta que las salidas de las mediciones y las simulaciones son cantidades aleatorias en las cuales la salida sería diferente cada vez que el experimento sea repetido. Esto hace necesario que al comparar dos alternativas, se tome en cuenta la variabilidad de los resultados. Simplemente comparar las medias de cada conjunto, puede llevar a conclusiones erróneas [22].

Existen diferentes formas de comparar diferentes alternativas, pero para el caso de dos sistemas con cargas de trabajo similares podemos usar técnicas como las observaciones de pares. Sin embargo, una vez que se ha terminado las pruebas, se debe interpretar los datos. Es necesario entender que el análisis solo produce resultados pero no conclusiones. Los resultados, que arroja el análisis, proveen una base sobre la cual se deben realizar las conclusiones. No obstante, diferentes personas que reciban el mismo conjunto de datos podrían obtener conclusiones diferentes y válidas [22].

4.1.10 PRESENTAR LOS RESULTADOS

El paso final de un análisis de rendimiento es comunicar los resultados. Los resultados son comunicados a otros interesados para poder tomar decisiones. Es

importante que los resultados se muestren de una manera que sea fácil de comprender. Esto, usualmente, implica que sean mostrados de una manera gráfica y sin una jerga relacionada a la “estadística” [22].

En algunos casos, donde los resultados no sean concluyentes, todo el proceso se debe realizar desde el principio, debe ser cíclico en vez de secuencial. Esto no significa que sea una falla, sino que son cambios en los pasos para poder tener resultados más concluyentes o más útiles para aquellos a los que son presentados los resultados [22].

En el caso del presente proyecto, los resultados servirán para la comparación con soluciones similares ya existentes. Lo que fue planteado como uno de los objetivos específicos del proyecto.

4.2 PRUEBAS DE RENDIMIENTO SOBRE EL SISTEMA

Para realizar las pruebas de rendimiento, necesitaremos seguir algunos pasos que incluyen la generación de los ejecutables del proyecto, así como la creación de un clúster. La técnica de evaluación seleccionada fue la medición de un sistema real (4.1.6). Esto significa que deberemos medir el tiempo de ejecución que toma procesar de acuerdo al experimento propuesto (4.1.8).

4.2.1 GENERACIÓN DE LOS EJECUTABLES DEL PROYECTO

Lo primero que se necesitará es generar ejecutables de la aplicación. Para esto se puede utilizar *stack* para compilar la librería y los ejecutables. El comando *stack install* compilará las dependencias de la aplicación y posteriormente generará los ejecutables de la aplicación y los moverá al directorio *$HOME*
.local
bin.

```
→  application git:(master) stack install
Copying from /Users/juanpaucar/projects/application/.stack-work/install/x86_64-osx/lts-10.8/
8.2.2/bin/mapper to /Users/juanpaucar/.local/bin/mapper
Copying from /Users/juanpaucar/projects/application/.stack-work/install/x86_64-osx/lts-10.8/
8.2.2/bin/reducer to /Users/juanpaucar/.local/bin/reducer
Copying from /Users/juanpaucar/projects/application/.stack-work/install/x86_64-osx/lts-10.8/
8.2.2/bin/reducer-id to /Users/juanpaucar/.local/bin/reducer-id

Copied executables to /Users/juanpaucar/.local/bin:
- mapper
- reducer
- reducer-id
```

Figura 4.1: Instalación de Stack

Esto genererará los tres ejecutables que tiene el proyecto. Estos ejecutables se corresponden a un ejecutable para el paso de “map” y dos para el proceso de “reduce”. Uno de estos simplemente pasará la información recibida por el STDIN del sistema y la escribirá al STDOUT. Este reducer es para poder medir simplemente el rendimiento con un reducer trivial.

A diferencia de lenguajes interpretados como Python o Ruby, que tienen interpretes instalados en distribuciones GNU/Linux por defecto, Haskell es un lenguaje compilado. Esto significa que los ejecutables generados contienen el código de máquina a ejecutar. En el caso de Python y Ruby, el código escrito en estos lenguajes deberá ser interpretado por los correspondientes paquetes en el sistema que se encargarán de hacer el uso de los paquetes del lenguaje que vienen en sus distribuciones por defecto.

Con cualquier lenguaje compilado se debe tener en cuenta como se enlanzan las librerías que usa el ejecutable. Existen dos maneras de enlazar librerías: estáticamente y dinámicamente. Si se genera el ejecutable con las librerías enlazadas dinámicamente, el mismo buscará esas librerías en el sistema que se ejecute. En el caso de Haskell, los ejecutables generados tienen enlazadas las librerías estáticamente. Esto significa que el ejecutable incluye las librerías necesarias para su ejecución. Sim embargo, sólo se empaqueta estáticamente las librerías de Haskell; en caso de que existieran dependencias en otro lenguaje como C, simplemente es necesario agregar unas opciones extras a la compilación.

```
$ stack build --ghc-options='-optl-static -optl-pthread
    '
```

No es algo que afecte a la aplicación creada debido a que no usa ninguna dependencia nativa escrita en C, pero es algo que se debe tener en cuenta cuando se crea ejecutables. Cabe aclarar que en el caso de lenguajes como Python y Ruby, también existen librerías que hacen referencia a librerías nativas. En ambos casos, tanto con lenguajes compilados como interpretados, es necesario instalar dichas dependencias en los nodos de Hadoop.

4.2.2 RECOPILACIÓN Y SUBIDA DE ARCHIVOS A S3

Antes de comenzar a crear el clúster, necesitamos subir los archivos que vamos a utilizar a S3, el servicio de almacenamiento de Amazon. Es necesario subirlo en S3 debido a que hace que no exista la configuración del sistema de archivos distribuido de Hadoop. En su lugar, AWS se encargará de mover tanto los ejecutables como el archivo a procesar a el sistema de archivos. La interfaz para subir archivos en S3 es bastante amigable tanto para añadir nuevos archivos como para navegar en los directorios que tiene S3.

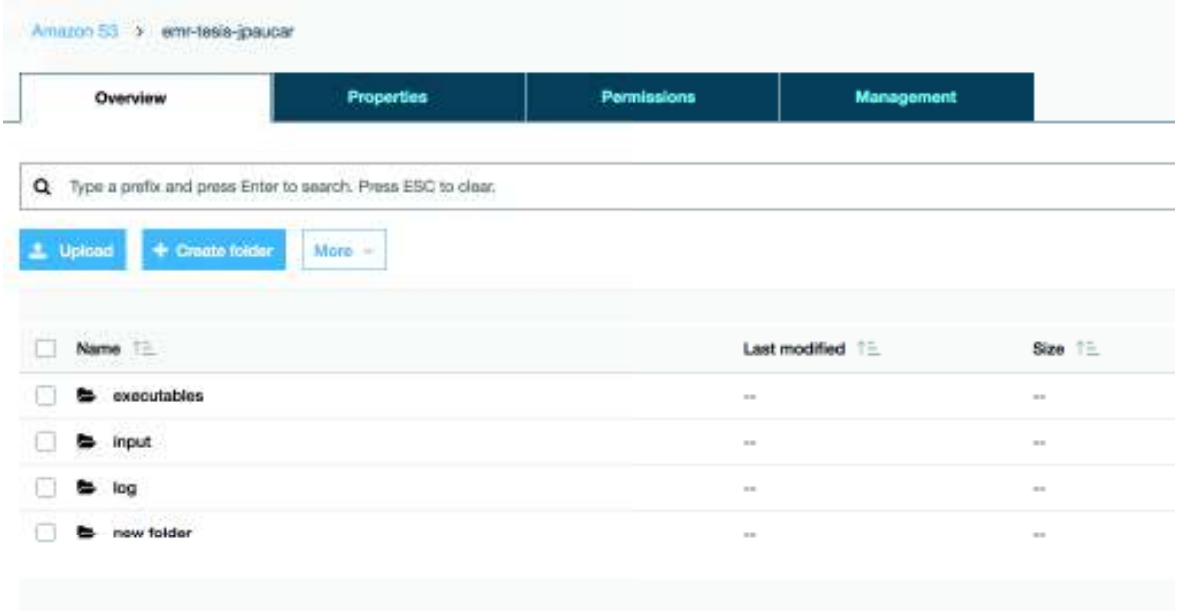

Figura 4.2: Subida de archivos a S3

4.2.3 CREACIÓN DE UN CLÚSTER

Una vez que se hayan subido los ejecutables y el archivo a procesar, se procede a crear un clúster en EMR (Elastic MapReduce). Cuando se entra al servicio se nos presenta una interfaz sencilla que explica el funcionamiento.

Figura 4.3: Interfaz de bienvenida de EMR

Cuando se comienza a crear un clúster de nodos en EMR, se tiene la “Configuración Rápida” y la “Configuración Avanzada”. En el caso de la “Configuración Avanzada” se puede seleccionar opciones cómo que servicios deben ser instalados con Hadoop. La aplicación relaizada está hecha para Hadoop Streaming por lo que la instalación por defecto es suficiente. Así también, se puede elegir si el clúster tendrá un registro de su actividad. Todo el registro estará disponible en S3 para posterior descarga.

Se puede crear un clúster completo que sólo se eliminará cuando esto se haga manualmente desde la consola de administración de EMR. Sin embargo, la otra opción de “Step execution” correrá los pasos que asignen y al terminar eliminará el clúster. Cabe recalcar que eliminar el clúster no afecta a los registros o “logs” que este haya generado así como tampoco a la salida del procesamiento del archivo. Estas salidas son guardadas en S3 y no dependen de la existencia del clúster.

Figura 4.4: Configuraciones generales de EMR

A continuación, se presenta el resto de opciones para configurar el clúster. Se tiene:

1. **Add steps:** Aquí se agregan los pasos que ejecutará el clúster así como el tipo de cada uno. Por ejemplo, podría primero ejeutar la aplicación desarrollada y posteriormente un map y un reduce que reciban la salida del paso anterior

2. **Software configuration:** Hadoop cuenta con aplicaciones extras que expanden su funcionalidad base. En este caso sólo es necesario que el Hadoop base esté disponible.

3. **Hardware configuration:** Aquí se encuentran las configuraciones que van a afectar los experimentos. El número de nodos en el clúster así como el tamaño de la instancias son especificados en esta sección.

4. **Security and access:** La configuración del clúster, aplicación y sistema de archivos, está a cargo de EMR. Sin embargo, existe la opción de conectarse a los nodos a través de SSH y tener un control más fino del clúster. Es por esto que el acceso a estos nodos está regido por permisos y reglas estándar en servidores de Amazon.

Figura 4.5: Otras configuraciones de EMR

Para agregar pasos adicionales, se desplagará otro menú con configuraciones específicas a los pasos seleccionados. Debido a que previamente seleccionado fue Haddop Streaming, el menú que se presenta contendrá las configuraciones apropiadas para esto.

1. **Step type:** El “tipo de paso” dependerá exclusivamente de lo que se haya seleccionado en configuraciones previas.

2. **Name:** Este nombre es útil en caso de tener múltiples pasos, para llevar un registro más claro de en que paso está.

3. **Mapper:** Esta configuración se refiere a una localización en un S3 donde se encuentra el ejecutable que será el “mapper”.

4. **Reducer:** Esta configuración se refiere a una localización en un S3 donde se encuentra el jecutable que será el “reducer”.

5. **Input S3 location:** Esta configuración se refiere a una localización en un S3 donde se encuentra el archivo que será procesado por el streaming.

6. **Output S3 location:** Los logs que genere la ejecución del paso pueden tener una localización distinta a los logs de la ejecución del clúster.

7. **Arguments:** Esta configuración se refiere tanto a parámetros que acepta Hadoop. En el caso de Hadoop Streaming, también se puede enviar parámetros de configuración a los ejecutables si fuese necesario.

8. **Action on failure:** Esta configuración nos permite definir que comportamiento tendrá este paso, si llegará a fallar en cualquier parte. En caso de falla nos permite elegir entre: Continuar descartando los datos que fallaron al procesarse, reiniciar el paso o cancelar el paso totalmente. Cabe recordar que Hadoop particiona el archivo de entrada, por lo que una falla en el procesamiento de alguno de las particiones no sigfinica que todo el paso haya fallado.

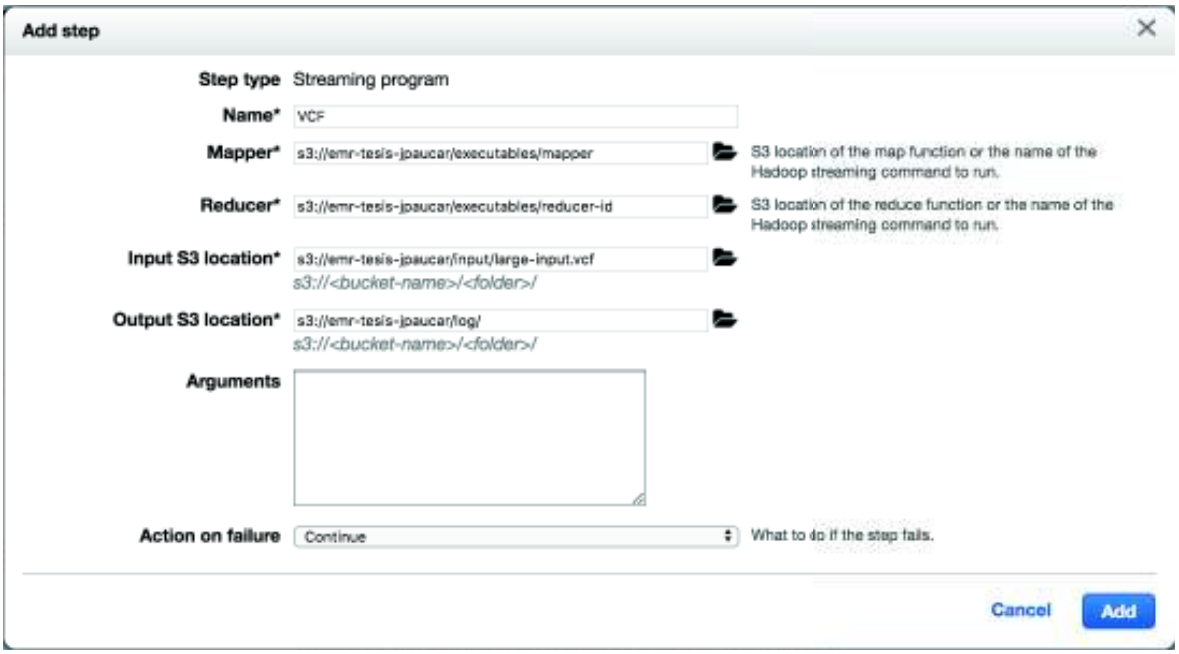

Figura 4.6: Configuraciones por cada paso de EMR

Tras la finalización de la ejecución de los pasos configurados, se tendrá la salida al procesamiento de cada paso, en donde se haya configurado su almancenamiento. Así también, se obtendrán los registros de la ejecución de Hadoop donde se encuentra la información que utilizaremos para el análisis de rendimiento.

4.2.4 SALIDAS DE LAS PRUEBAS DE RENDIMIENTO

Tras finalizar las pruebas de rendimiento se procede a acumular estos datos en tablas en el apendice .1. Cada tabla contiene la información de las mediciones de la ejecución del experimento variando los nivdeles de los factores descritos en la tabla con los parámetros y sus respectivos niveles de la sección 4.1.4.

4.3 ANÁLISIS DE LOS RESULTADOS

Los datos tomados de la medición de tiempo en el experimento nos permiten desglosar la ecuación inicial de tiempo respecto a los factores.

Los factores que se variaron fueron: p cómo el número de nodos y T el tamaño de la instancia. Esto significa que el tiempo está en función de estas dos variables. Todavía, se tiene W como una constante con el valor del tamaño del archivo procesado [5].

$$t_{total} = f(W, p, T) \tag{4.3}$$

Al ser W una constante se simplifica a:

$$t_{total} = f(p, T) \tag{4.4}$$

Así también, se puede descomponer al tiempo total de ejecución como la suma de los pasos principales de una aplicación MapReduce: “map”, “shuffle” y “reduce” [5].

$$t_{total} = t_{map} + t_{shuffle} + t_{reduce} \tag{4.5}$$

De esto, podemos descomponer las ecuaciones de tiempo para “map” y “reduce”. El caso de “shuffle” es algo diferente y será descompuesto posteriormente. A pesar de que W es constante, nos sirve para derivar algunas ecuaciones que serán útiles

para el análisis de rendimiento. Cabe recordar que Hadoop divide las tareas en trabajos más pequeños que luegos distribuye a los nodos [5].

Sea:

- w_{chunk} = El tamaño de chunk de Hadoop
- n_{map} = El número de operaciones map

$$n_{map} = \left\lceil \frac{W}{w_{chunk}} \right\rceil \tag{4.6}$$

El valor de W es constante y el valor de w_{chunk} tiene un valor por defecto de 128MB pero es configurable. Además conocemos el valor de t_{map} que es una de las mediciones durante los experimentos [5].

Sea:

- t_{umap} = El tiempo unitario de una tarea map

$$t_{map} = t_{umap} \times n_{map} \times \left\lceil \frac{1}{p} \right\rceil \tag{4.7}$$

$$t_{umap} = \frac{t_{map}}{\left\lceil \frac{n_{map}}{p} \right\rceil} \tag{4.8}$$

Además de la ecuación previa, también se puede expresar el tiempo de una tarea de map en función de C cómo el costo computacional de la aplicación y T cómo el tamaño de la instancia que ejecuta la aplicación [5].

$$t_{umap}[s] = \frac{w_{chunk}[byte] \times C[\frac{flop}{byte}]}{T[\frac{flop}{s}]} \tag{4.9}$$

Debido a eso se puede caracterizar la aplicación con una función de $\frac{C}{T}$. La variable C es el costo computacional de la aplicación y este no varía entre experimentos, debido a que es la misma aplicación corriendo en diferentes entornos [5]. Sin embargo, T es el tamaño de las instancias donde se ejecutaba la aplicación. Entonces obtenemos:

$$\frac{C}{T} = \frac{t_{umap}}{w_{chunk}} \tag{4.10}$$

En este caso, se tomará en cuenta sólo al tiempo unitario de la tarea “map”. Debido a que la tarea “reduce” que se utilizó en los experimentos es trivial. Por lo que se considerará este igualdad:

$$t_{umap} = t_{uT} \tag{4.11}$$

Partiendo de ecuaciones anteriores, para los dos tamaños de instancia que se usó, se tienen:

$$\frac{C}{T_{large}} = \frac{t_{ularge}}{w_{chunk}} \tag{4.12}$$

$$\frac{C}{T_{medium}} = \frac{t_{umedium}}{w_{chunk}} \tag{4.13}$$

C tiene el mismo valor en ambas ecuaciones. Esto implica que al poner ambas ecuador en función de dicha variable, podemos reemplazar en la igualdad y eliminar el tamaño del “chunk”:

$$\begin{aligned} \frac{C}{T_{large}} &= \frac{t_{ularge}}{w_{chunk}} \\ \frac{C}{T_{medium}} &= \frac{t_{umedium}}{w_{chunk}} \end{aligned} \tag{4.14}$$

$$\begin{aligned} C &= \frac{t_{ularge} \times T_{large}}{w_{chunk}} \\ C &= \frac{t_{umedium} \times T_{medium}}{w_{chunk}} \end{aligned} \tag{4.15}$$

$$\frac{t_{umedium} \times T_{medium}}{w_{chunk}} = \frac{t_{ularge} \times T_{large}}{w_{chunk}} \tag{4.16}$$

Finalmente obtendremos una relacion entre los dos niveles de T que hemos usado en los experimentos.

$$t_{umedium} \times T_{medium} = t_{ularge} \times T_{large} \tag{4.17}$$

Tanto $t_{umedium}$ como t_{ularge} son valores conocidos debido a los experimentos. Para completar la relación usaremos el promedio en las mediciones tanto del uno cómo del otro. Pero dichos valores corresponden solamente a las pruebas con un nodo, como se ve en la tabla 1.

$$\begin{aligned} 4{,}721801446 \times T_{medium} &= 3{,}077067815 \times T_{large} \\ T_{large} &= 1{,}54 \times T_{medium} \end{aligned} \tag{4.18}$$

Con dicha ecuación tenemos una relación entre los valores de T de una instancia de tamaño $m1.medium$ y $m4.large$. Usando esa relación como punto de partida se puede hacer predicciones acerca de cual será el tiempo requerido si se elije un tamaño de la instancia y otro.

4.3.1 DIFERENCIAS ENTRE EL NÚMERO DE NODOS

En los anexos, se encuentran los valores de las mediciones con cada uno de los niveles de los factores escogidos .1. Para la ecuación anterior, se tomó en cuenta el valor de tiempo unitario por cada tarea de map. Este debería ser casi constante, a pesar del número de nodos que se agreguen.

El tiempo unitario de una tarea de map se puede expresar en función del número de tareas map y el número de nodos en el clúster. El número de tareas map no va a variar debido a que ese valor únicamente depende del tamaño de la carga. Entonces se tiene:

$$t_{umap} = \frac{t_{map}}{\left\lceil \frac{n_{map}}{p} \right\rceil} \tag{4.19}$$

El tiempo que toma terminar todas las tareas map se expresa como t_{map}. Esto significa que dependerá de las mediciones de tiempo que se haya hecho. Sin embargo,

Tabla 4.2: Tiempo por tarea unitario de map en cada tamaño de instancia

Número de nodos (p)/ Tamaño instancia (T)	**m4**	**m1 medido**	**m1 esperado**	**% de tiempo extra**
1	3.08	4.72	4.72	0.00
2	3.21	5.89	4.93	19.43
3	3.82	6.35	5.86	8.40
4	4.05	6.36	6.21	2.45

la relación de $T_{large} = 1{,}54 \times T_{medium}$ es una medición si esperamos que toda la aplicación sea trivialmente paralelizable en la tarea de map.

Se puede apreciar que los valores que se esperan la relación de T para un sólo nodo, son mayores hasta por $19{,}43\,\%$. Esto es debido a que a pesar de el número trabajos map que realiza un nodo se divide para el número total de nodos, todavía existe un nodo maestro que también dedica parte de sus recursos a la orquestación de la distribución de trabajo. Se puede apreciar una tendencia a la reducción del tiempo que se espera.

Esto significa que la relación entre T_{large} y T_{medium} varía de acuerdo a número de nodos. Entonces tenemos las siguientes relaciones entre tamaños de instancia:

$$p = 1 T_{large} = 1{,}54 \times T_{medium} \tag{4.20}$$

$$p = 2 T_{large} = 1{,}83 \times T_{medium} \tag{4.21}$$

$$p = 3 T_{large} = 1{,}66 \times T_{medium} \tag{4.22}$$

$$p = 4 T_{large} = 1{,}57 \times T_{medium} \tag{4.23}$$

Si recordamos, tiempo estaba en función del número de nodos y el tamaño de la

instancia $t = f(T, p)$. Con las relaciones entre los tamaños de T dependiendo de p, ya se puede predecir el tiempo que tomará la ejecución de la aplicación. Idealmente, la relación entre los tamaños de instancia no debería variar, sin embargo parece que se estabiliza cuando se aumenta más el número de instancias.

4.4 COMPARACIÓN CON OTRAS SOLUCIONES

Tal como se estableció en los objetivos de este proyecto, se compara la aplicación desarrollada frente a otras soluciones ya existentes para poder obtener una idea del rendimiento de la aplicación desarrollada.

4.4.1 CONSIDERACIONES PREVIAS

En la tabla de comparación de características con soluciones ya existentes 2.3, se pudo observar que varias herramientas pueden procesar archivos VCF. Para que la comparación sea justa. En el experimento se usó un “mapper” que procesa líneas de un archivo VCF y que los convierte en una estructura de datos con lógica para la aplicación. Sin embargo, el “reducer” que se usó para el experimento es uno trivial que simplemente escribirá al STDOUT lo que lee del STDIN.

Es decir, la comparación entre todas las aplicaciones involucra el procesamiento de los archivos VCF. Se elige sólamente procesar los archivos debido a que una vez que son interpretados como una estructura con sentido para la aplicación, se puede proceder a hacer cualquier otro paso o aplicación. Además, el procesamiento del archivo es la mínima operación común entre todas las herramientas. Hacer algo más avanzado implicaría repetir el desarrollo de la aplicación para que se ajuste al ambiente de cada una de las demás aplicaciones.

Ninguna de las aplicaciones listadas en la tabla 2.3 funciona sobre Hadoop. Esto signifca que las aplicaciones no correrán en el servicio EMR (Elastic MapReduce) de Amazon, sino que deberán correr en un servicio diferente que todavía nos permite mantener los factores que se tuvieron en cuenta en el experimento. Este servicio

Tabla 4.3: Comparación de los tiempos promedio en segundos de procesamiento de las soluciones

Tamaño de la instancia\ Software	**Annovar**	**VEP**	**SnpEff**	**Propia**
m1	4466.29	5415.47	5529.25	5807.56
m4	3021.86	3386.58	3449.46	3674.99

es EC2 (Elastic Cloud Computing).

Al no ser aplicaciones que puedan correr sobre Hadoop, estas no pueden paralelizarse simplemente al dividir la entrada en pedazos. Por lo que el factor p que es el número de nodos que corren la aplicación, siempre será 1 y pasa a ser una constante al igual que el archivo de entrada W. Esto significa que la ecuación se verá grandemente simplificada al solamente poder variar el tamaño de la instancia sobre la que se ejecuta.

$$\begin{aligned} t &= f(W, p, T) \\ t &= f(T) \end{aligned} \tag{4.24}$$

Esto significa que los resultados de tiempo total serán siempre analizados contra la ejecución en un sólo node de Hadoop.

4.4.2 RESULTADOS FRENTE A OTRAS SOLUCIONES

Se realizó varias mediciones del tiempo que toma la ejecución de las aplicaciones con cada una de las aplicaciones que se seleccionaron. El tiempo se encuentra solamente en función del tamaño de la instancia que lo ejecuta. A continuación, se lista el tiempo promedio del procesamiento del archivo VCF.

Las actuales soluciones parecen tener tiempos de ejecución en promedio más cortos. La más rápida parece ser Annovar y la más lenta, en promedio, es la solución desarrollada en el presente proyecto.

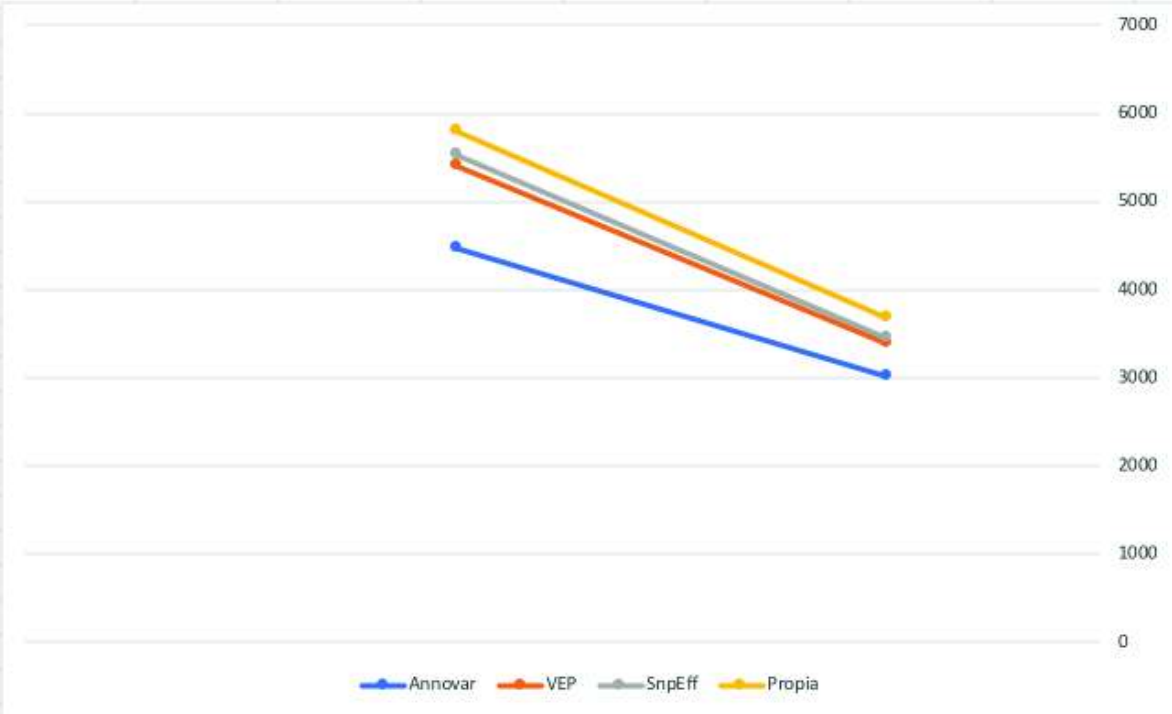

Figura 4.7: Promedios de tiempo de procesamiento en segundos (menos es mejor)

4.4.3 ANÁLISIS DE LOS RESULTADOS

En las mediciones realizadas, todas las herramientas realizadas actuales parecen poder procesar archivos VCF más rápido. Sin embargo, como se mencionó en las consideraciones previas hay más aspectos a considerar que simplemente la velocidad de ejecución con un sólo factor como es el tamaño de la instancia.

1. Paralelización: Cómo se menciona en las justificaciones para hacer una aplicación que trabje sobre Hadoop. Este framework se encarga de orquestar el procesamiento paralelo de información requiriendo mínima intervención. Esto significa que casi se puede dividir el tiempo que toma en un nodo y dividirlo para los nodos que se deseen agregar. En resumen, la aplicación desarrollada goza de una paralelización en el procesamiento de datos que ninguna otra de las herramientas comparadas puede ofrecer.

2. Costo: También se mencionó previamente que las aplicaciones no iban a poder ser ejecutadas en el mismo servicio de Amazon, debido a que EMR es

exclusivo de aplicaciones que funcionen sobre Hadoop o una de sus herramientas. A pesar de que EMR internamente usa las mismas máquinas que EC2, el costo no es el mismo. Es una diferencia en el modelo de negocio, donde EC2 está hecha para funcionar como un servidor de propóstio general. EMR, por otro lado sólo se provisiona con las herramientas de Hadoop; además, es común que los servidores sean “destruidos” una vez que se acabó de procesar datos en Hadoop. Esto significa que los mismos vuelven a estar disponibles para otro cliente de AWS creando un clúster.

La diferencia en costes es lo suficientemente grande para que una aplicación que se ejecute en EMR sea muchas veces más económica. Por ejemplo, el costo de alquiler de las instancias en EC2 (Región Oregon EEUU) es:

- $m1.medium$: 0.087 USD la hora
- $m4.large$: 0.10 USD la hora

Por otro lado, el coste de el mismo tamaño de instancias en EMR, dentro de la misma región, es:

- $m1.medium$: 0.022 USD la hora
- $m4.large$: 0.030 USD la hora

El precio de la misma instancia en EMR es una fracción del costo que tendría en EC2. Pero, también se debe considerar que en EMR existe un cargo adicional por almacenar los archivos en S3. El mismo que dependerá de que volumen de datos se mueva y el nivel de disponibilidad del archivo. Sin embargo, el costo estimado por experimento en el almacenamiento fue de 0.007 USD. Hay que mencionar que no es obligatorio almacenar los archivos a procesar en S3. Pero, para el experimento se hizo de esta manera que no requiere configurar el sistema de archivos distribuido de Hadoop.

De los datos presentados, podemos concluir que la aplicación desarrollada tiene un tiempo de ejecución mayor a las soluciones que existen actualmente para procesar archivos VCF. Lo que significa que el rendimiento es menor para procesar archivos.

Sin embargo, se estaría dejando fuera a otros factores como el paralelizamiento del proceso que puede reducir tiempos. Además, en entornos como la nube, el tiempo y costo que tiene realizar el procesamiento de datos son especialmente críticos cuando se hace a gran escala. Acciones como la configuración manual de muchos servicios o dependencias pueden convertirse en un gasto innecesario de recursos. Pero, dependerá de cada proyecto realizar el cálculo del costo estimado en diferentes escenarios. Es ahí donde es especialmente útil la Planeación de Capacidad y el Análisis de Rendimiento de un sistema.

CAPÍTULO 5. CONCLUSIONES Y RECOMENDACIONES

5.1 CONCLUSIONES

- Un algoritmo desarrollado para MapReduce debe tener en cuenta que puede lidiar con cualquier parte de todo el conjunto de datos de entrada, esto significa que debe poder ser aplicable en cualquier elemento del conjunto de entrada.
- El modelo de programación MapReduce permite el desarrollo de una aplicación para la cual no es relevante la manera u orden en que se procesa los datos. Esto hace que el procesamiento de grandes cantidades de datos sea bastante más trivial que al usar otro modelo de programación.
- El genoma humano consta de una gran cantidad de datos que, a pesar de beneficiarse del aumento de poder de computación, todavía toma una considerable cantidad de tiempo para procesar. Esta gran cantidad de datos se beneficia de poder ser procesa de una manera paralela cuando sea posible. Lo que significa que el procesamiento de variables es un candidato perfecto para MapReduce.
- La reducción progresiva en los costos de computación en la nube se traduce en una reducción directa de los costos del procesamiento de grandes cantidades de datos de la escala que se necesita, para los diferentes pasos que componen al secuenciamiento del genoma.
- El uso del framework como Hadoop le da una ventaja a la aplicación desarrollada en el presente proyecto respecto a la paralelización de trabajo. Esto significa que la aplicación desarrollada puede escalar tanto verticalmente

(agregando más recursos a los servidores que ejecutan la aplicación) como horizontalmente (agregando más servidores para ejecutar la aplicación de manera conjunta). A diferencia de otras soluciones que, por su diseño y arquitectura, están hechas para escalar únicamente de manera vertical.

- El uso de un lenguaje funcional hizo que el desarrollo, de una herramienta capaz de procesar archivos VCF, sea sencillo debido a las herramientas inherentes a la arquitectura del lenguaje de programación como los *Parser Combinators*. Así también, el modelo de evaluación perezosa de datos permite manejar una gran cantidad de datos con mayor facilidad que la evaluación por adelantado que tienen la mayoría de los lenguajes de programación.

- El uso del desarrollo guiado por pruebas ayuda en el diseño de una aplicación más simple y desacoplada. Debido al uso de la programación funcional, el desarrollo guiado por pruebas no aporta al diseño de la aplicación puesto que, al usarse únicamente funciones, se evita el acoplamiento de partes del código. De todas maneras, es útil para tener una verificación de que la aplicación está funcionando de manera esperada.

- La generación de un modelo de rendimiento ayuda a la obtención de un cálculo aproximado del tiempo que tomará la ejecución de la aplicación. Esto se puede relacionar con el costo estimado que tiene la ejecución de dicha aplicación en un ambiente de computación en la nube. Sin embargo, algunos proveedores de servicios de computación en la nube tienen más métricas para el cálculo de estos costos, además de solamente el tiempo de uso de sus plataformas, por lo que dichos costos deberían ser tomados en cuenta.

- La aplicación desarrollada únicamente cubre el análisis de los efectos de las variaciones mediante el uso de un API externa o una base de conocimiento local. Por lo que no es necesario cubrir partes previas del proceso de análisis para lograr cubrir esta etapa.

- Los lenguajes funcionales son declarativos y no imperativos como los lenguajes orientados a objetos. Esto significa que durante el desarrollo de la aplica-

ción se piensa en el qué se debe hacer y no en el cómo se debe hacer. Esto hace que el desarrollo ocurra en una abstracción de más alto nivel.

- La aplicación desarrollada, al usar el modelo de programación MapReduce, está dividida en dos partes. El *mapper* se encarga del procesamiento de la entrada convirtiéndola en estructuras con lógica y el *reducer* se encarga de detectar el efecto de la variación y si este afecta a la salud.

- Debido a que Scrum es un framework, este permite la selección de ciertas partes del mismo que mejor se ajusten para el desarrollo de cada proyecto. Dado que la aplicación fue desarrollada por una sola persona, se pudo obviar muchas partes que, de otra manera, hubiesen causado un entorpecimiento innecesario del desarrollo.

5.2 RECOMENDACIONES

- Algunos, trabajos futuros podrían comenzar a mudar más fases del secuenciamiento del genoma hacia algoritmos MapReduce que sean compatibles con Hadoop, de manera que el procesamiento pueda escalar horizontalmente.

- Hadoop cuenta con muchas herramientas que funcionan junto con él o como reemplazo a tareas muy específicas del mismo. Un caso particular es Spark que procesa flujos de datos. El uso de Spark pudo ser beneficioso también en este proyecto debido a que tiene soporte nativo de Scala y se beneficia enormemente de que Scala sea un lenguaje que soporte la Programación Funcional y la Programación Orientada a Objetos. Recomendaría que trabajos futuros hagan uso de las herramientas que mejor se ajusten a su caso.

- La librería, para el procesamiento de archivos VCF, que fue desarrollada como parte de este proyecto se puede beneficiar de un ajuste en el rendimiento. Existe oportunidad de mejora tanto en la representación de las variaciones como en la manera en que procesan los datos. Trabajos futuros podrían expandirla para cubrir más escenarios del secuenciamiento.

- Posibles futuros proyectos podrían desarrollar aplicaciones prácticas, para usuarios finales, que usen la librería desarrollada para el procesamiento de archivos VCF.

- A pesar de la flexibilidad de SCRUM como framework para el desarrollo de software, en el caso de un equipo pequeño puede que una mezcla de ciertas partes de SCRUM con Extreme Programming sean la mejor solución. Esto debido a que Extreme Programming está hecho para equipos pequeños, pero no cubre ciertas herramientas que tiene SCRUM. Recomendaría, que futuros proyectos elijan la combinación que más favorezca al desarrollo de la aplicación.

- Aprender un nuevo paradigma de programación implica tener una nueva forma de pensar soluciones para distintos problemas. Trabajos futuros podrían considerar usar la Programación Funcional para el desarrollo de nuevas aplicaciones.

GLOSARIO

ADN Molécula compleja que contiene toda la información necesaria para construir y mantener un organismo, todos los seres vivos la poseen en sus células[6]. 3–7, 20, 22

genoma Es el conjunto completo de ADN de un organismo, que incluye todos sus genes. Contiene la información completa para construir y mantener ese organismo [15]. 11, 20

genotipo Es la identidad genética hereditaria completa, es un genoma único personal. También se refiere a un conjunto partículas de genes llevados por un individuo [30]. 11

Hadoop Implementación del modelo de programación MapReduce hecha por Apache. 6–9, 11–13, 15, 21, 45, 47, 58

Haskell Lenguaje de programación basado en el cálculo lambda tipado, con las siguientes características: estáticamente tipado, puramente funcional, con inferencia de tipos, concurrente, con evaluación perezosa declarativo[17]. 8, 9, 13, 21, 23–26, 34, 44, 45

MapReduce Es un modelo de programación desarrollado por Google para el procesamiento de grandes cantidades de datos de una manera paralela [23]. 15, 21, 24, 31, 48

SCRUM Marco de trabajo para el desarrollo de proyectos de software con un enfoque en metologías ágiles [33]. 27, 28, 30

SNPs *Single nucleotide polymorphisms* o polimorfismos de un sólo nuclétido. El ADN está compuesto de cuatro nucleótidos base que se combinan. Los SNPs son el tipo de variación genética más común entre las personas. Cada SNP representa una diferencia entre un sólo bloque de ADN. Por ejemeplo, en lugar de tener un nuclétido C (citocina) se encuentra el nucleótido T (timina). Los SNPs ocurren a lo largo de todo el ADN de una persona. En promedio, ocurren una vez cada 300 nucleótidos, lo que significa que existen aproximadamente 10 millones de SNPs en el genoma humano. Dependiendo de la región del ADN en donde se ecnuentren pueden tomar un rol más directo en enfermedades. [25]. 16, 17

VCF Variant Call Format. 3, 6, 11–16, 19, 20, 26, 28, 31, 38, 39, 41, 42, 46, 47, 49, 80–82, 84

BIBLIOGRAFÍA

[1] ANNOVAR. *ANNOVAR Documentation*. 2016. URL: http://annovar.openbioinformatics.org/en/latest/.

[2] Doaitse Swierstra Bastiaan Heeren Jurriaan Hage. «Generalizing Hindley-Milner Type Inference Algorithms». En: *technical report UU-CS-2002-031* (2002).

[3] Kent Beck. *Test-Driven Development by Example*. Addison-Wesley Professional, 2003.

[4] Claudia Calabrese y col. «MToolBox: a highly automated pipeline for heteroplasmy annotation and prioritization analysis of human mitochondrial variants in high-throughput sequencing». En: *Bioinformatics* 30.21 (nov. de 2014), págs. 3115-3117. DOI: 10.1093/bioinformatics/btu483. URL: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC4201154/.

[5] Iván Carrera. «Performance Modeling of MapReduce Applications for the Cloud». En: *Biblioteca de la UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL* (2014).

[6] Heidi Chial y col. *Essentials of Genetics*. Nature Education, Enero, 2014.

[7] P. Cingolani y col. «A program for annotating and predicting the effects of single nucleotide polymorphisms, SnpEff: SNPs in the genome of Drosophila melanogaster strain w1118; iso-2; iso-3». En: *Fly* 6.2 (2012), págs. 80-92.

[8] International Human Genome Sequencing Consortium. «Initial sequencing and analysis of the human genome». En: *Nature* 409.6822 (feb. de 2001), págs. 860-921. URL: http://dx.doi.org/10.1038/35057062.

[9] Defense Deparment of the United States of America. *Systems Engineering Fundamentals*. Defense Acquisition University Press, 2001.

[10] Ensembl. *Variant Effect Predictor*. URL: http://www.ensembl.org/info/docs/tools/vep/index.html.

[11] Neal Ford. *Functional Thinking*. O'Reilly Media, 2014.

[12] The Apache Software Foundation. *Welcome to Apache™ Hadoop®!* URL: https://hadoop.apache.org/.

[13] Martin Fowler. *Refactoring: Improving the Design of Existing Code*. Addison Wesley Professional, 2012.

[14] W Florian Fricke y David A Rasko. «Bacterial genome sequencing in the clinic: bioinformatic challenges and solutions». En: *Nature Reviews Genetics* 15.1 (2014), págs. 49-55.

[15] Genetics Home Reference. *What is a genome?* Mayo de 2016. URL: https://ghr.nlm.nih.gov/primer/hgp/genome.

[16] Global Alliance for Genomics and Health Data Working group. *The Variant Call Format (VCF) Version 4.2 Specification*. Ene. de 2015. URL: https://samtools.github.io/hts-specs/VCFv4.2.pdf.

[17] haskell.org. *Haskell*. 2016. URL: https://www.haskell.org/.

[18] Andrew SCHNABEL; Peter BEERLI; Arnaud ESTOUP; David HILLIS. «A guide to software packages for data analysis in molecular ecology». En: (1998).

[19] John Hughes. «Why Functional Programming Matters». En: *Research Topics in Functional Programming* (1990).

[20] Illumina. *An Introduction to Next-Generation Sequencing Technology*. Inf. téc. Illumina, 2016.

[21] National Human Genome Research Institute. *Frequently Asked Questions About Genetic Testing*. Ago. de 2015. URL: http://www.genome.gov/19516567.

[22] Raj Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley, abr. de 1991.

[23] Ralf Lämmel. «Google's MapReduce programming model – Revisited». En: *Science of Computer Programming* (2007).

[24] Miran Lipovača. *Learn you a Haskell for great good!* No Starch Press, 2011.

[25] National Institute of Health. *What are single nucleotide polymorphisms (SNPs)?*

[26] NCBI. *GenBank Overview*. 2015. URL: http://www.ncbi.nlm.nih.gov/genbank/ (visitado 2015).

[27] Bryan O'Sullivan. «What's in a parser? Attoparsec rewired». En: (2010).

[28] Bryan O'Sullivan, Don Stewart y John Goerzen. *Real World Haskell*. O'Reilly Media, 2008.

[29] Stephan Pabinger y col. «A survey of tools for variant analysis of next-generation genome sequencing data». En: *BRIEFINGS IN BIOINFORMATICS* 15.2 (ene. de 2013), págs. 256-278.

[30] Personal Genetics Education Project. *What is genotype? What is phenotype?* URL: http://www.pged.org/what-is-genotype-what-is-phenotype/.

[31] Genetics Home Reference. *What are single nucleotide polymorphisms (SNPs)?* Mayo de 2016. URL: https://ghr.nlm.nih.gov/primer/genomicresearch/snp.

[32] Genetics Home Reference. *What is a genome?* 2016. URL: http://ghr.nlm.nih.gov/handbook/hgp/genome.

[33] Ken Schwaber y Jeff Sutherland. *The Definitive Guide to Scrum: The Rules of the Game*.

[34] solarwinds. «TOP 4 CAUSES OF STORAGE I/O BOTTLENECKS AND HOW TO MITIGATE THEM». En: (2015).

[35] Lincoln D Stein. «The case for cloud computing in genome informatics». En: *Genome Biology* (2010).

[36] Zachary D Stephens y col. «Big Data: Astronomical or Genomical?» En: *PLoS Biol* 13.7 (2015), e1002195.

[37] The Agile Dictionary. *Spyke*. 2018. URL: http://agiledictionary.com/209/spike/.

[38] Gudula Rünger Thomas Rauber. *Parallel Programming for Multicore and Cluster Systems*. 3.ª ed. Springer, 2013.

[39] Hiroshi Wada, Junichi Susuki y Oba Katsuya. «Modeling Non-Functional Aspects in Service Oriented Architecture». En: *2006 IEEE International Conference on Services Computing (SCC'06)* (2006).

[40] Kris Wetterstrand. *DNA Sequencing Costs: Data from the NHGRI Genome Sequencing Program (GSP)*. Dic. de 2017. URL: http://www.genome.gov/sequencingcosts/.

[41] Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, 2015.

[42] Brent Yorgey. *Typeclassopedia*. URL: https://wiki.haskell.org/Typeclassopedia.

ANEXOS

.1 TABLAS DE RESULTADOS DE LAS MEDICIONES

.1.1 MEDICIONES EN M1.MEDIUM

Tabla 1: Tiempo en segundos para 1 nodo en m1.medium

map	shuffle	reduce	total	nmap	tumap
4453.6	34.31	1275.48	5763.39	960	4.639166667
4581.28	34.31	1166.51	5782.1	960	4.772166667
4601.62	32.68	1144.02	5778.32	960	4.793354167
4334.73	32.52	1279.3	5646.55	960	4.51534375
4592.28	32.26	1155.51	5780.05	960	4.783625
4565.02	30.15	1392.51	5987.68	960	4.755229167
4646.76	34.07	1135.04	5815.87	960	4.840375
4513.89	30.91	1381.21	5926.01	960	4.70196875
4484.78	30.64	1117.48	5632.9	960	4.671645833
4694.22	33.15	1220.91	5948.28	960	4.8898125
4520.58	31.64	1275.97	5828.19	960	4.7089375
4644.12	31.17	1275.59	5950.88	960	4.837625
4550.35	32.76	1148.23	5731.34	960	4.739947917
4401.11	34.14	1300.44	5735.69	960	4.584489583
4459.39	33.76	1313.45	5806.6	960	4.645197917

Continúa en la siguiente página

Tabla 1 – Continación de la página anterior

map	shuffle	reduce	total	nmap	tumap
4361.22	30.66	1254.24	5646.12	960	4.5429375
4656	30.77	1193.7	5880.47	960	4.85
4347.18	28.42	1296.37	5671.97	960	4.5283125
4589.54	31.66	1177.55	5798.75	960	4.780770833
4556.17	34.3	1253.91	5844.38	960	4.746010417
4416.86	34.3	1357.1	5808.26	960	4.600895833
4571.48	30.61	1289.73	5891.82	960	4.761958333
4648.17	29.47	1183.99	5861.63	960	4.84184375
4362.15	29.93	1229.69	5621.77	960	4.54390625
4518.97	29.63	1299.45	5848.05	960	4.707260417
4565.01	29.05	1174.46	5768.52	960	4.75521875
4538.72	33.51	1234.24	5806.47	960	4.727833333
4654.21	34.22	1221.28	5909.71	960	4.848135417
4618.78	35.58	1327.52	5981.88	960	4.811229167
4549.85	28.8	1124.59	5703.24	960	4.739427083
4302.03	28.87	1315.19	5646.09	960	4.48128125
4659.83	34.91	1251.4	5946.14	960	4.853989583
4395.19	31.96	1200.02	5627.17	960	4.578322917
4499.63	33.74	1263.7	5797.07	960	4.687114583
4757.03	33.04	1161.22	5951.29	960	4.955239583
4510.14	31.95	1191.98	5734.07	960	4.6980625
4658.79	35.3	1240.62	5934.71	960	4.85290625
4495.41	31.28	1281.17	5807.86	960	4.68271875
4513.56	29.25	1185.03	5727.84	960	4.701625
4438.71	32.1	1309.57	5780.38	960	4.62365625
4580.27	32.26	1317.06	5929.59	960	4.771114583
4483.15	34.94	1312.1	5830.19	960	4.669947917
4529.47	30.92	1274.89	5835.28	960	4.718197917

Continúa en la siguiente página

Tabla 1 – Continación de la página anterior

map	shuffle	reduce	total	nmap	tumap
4485.25	32.66	1304.29	5822.2	960	4.672135417
4640.43	30.23	1252.46	5923.12	960	4.83378125
4705.78	32.44	1221.84	5960.06	960	4.901854167
4540.92	32.59	1294.59	5868.1	960	4.730125
4512.94	30.81	1139.13	5682.88	960	4.700979167
4406.97	31.96	1170.43	5609.36	960	4.59059375
		Promedio	5807.556939		4.721801446
		Desviación	105.7793084		0.108320153

Tabla 2: Tiempo en segundos para 2 nodos en m1.medium

map	shuffle	reduce	total	nmap	tumap
2686.03	46.33	799.8	3532.16	480	5.595895833
2886.15	51.65	827.59	3765.39	480	6.0128125
2902.99	47.51	695.83	3646.33	480	6.047895833
2810.74	47.24	753.95	3611.93	480	5.855708333
2875.06	51.89	810.07	3737.02	480	5.989708333
2736.93	49.48	748.51	3534.92	480	5.7019375
2783.91	49.67	788.6	3622.18	480	5.7998125
2830.09	48.6	706.09	3584.78	480	5.896020833
2912.52	49.98	786.07	3748.57	480	6.06775
2863.95	48.99	753.51	3666.45	480	5.9665625
2787.8	47.18	758.31	3593.29	480	5.807916667
2813.79	47.79	778.49	3640.07	480	5.8620625
2722.25	46.98	811.28	3580.51	480	5.671354167
2865.48	51.05	841.11	3757.64	480	5.96975

Continúa en la siguiente página

Tabla 2 – Continación de la página anterior

map	shuffle	reduce	total	nmap	tumap
2803.77	46.43	690.09	3540.29	480	5.8411875
2950.43	49.49	688.25	3688.17	480	6.146729167
2845.71	50.23	772.97	3668.91	480	5.9285625
3009.42	49.69	716.45	3775.56	480	6.269625
2855.54	48.39	747.56	3651.49	480	5.949041667
2791.84	46.34	680.34	3518.52	480	5.816333333
2796.24	50.41	764.72	3611.37	480	5.8255
2664.4	46.02	791.76	3502.18	480	5.550833333
2728.35	45.99	759.47	3533.81	480	5.6840625
2756.68	47.52	806.83	3611.03	480	5.743083333
2864.33	47.86	696.6	3608.79	480	5.967354167
2839.25	47.23	716.62	3603.1	480	5.915104167
2837.15	50.1	722.22	3609.47	480	5.910729167
2938.17	51.21	724.24	3713.62	480	6.1211875
2718.89	47.3	725.72	3491.91	480	5.664354167
2810.99	46.32	679.46	3536.77	480	5.856229167
2982.47	50.91	697.1	3730.48	480	6.213479167
2785.67	49.36	729.24	3564.27	480	5.803479167
2905.98	50.22	770.68	3726.88	480	6.054125
2913.02	48.86	696.92	3658.8	480	6.068791667
2793.81	48.32	650.62	3492.75	480	5.8204375
2888.55	48.95	680.81	3618.31	480	6.0178125
2820.08	47.53	776.3	3643.91	480	5.875166667
2751.21	45.54	701.01	3497.76	480	5.7316875
2931.39	48.41	697.27	3677.07	480	6.1070625
2779.15	46.17	663.89	3489.21	480	5.789895833
2771.65	48.66	756.13	3576.44	480	5.774270833
2711.13	47.97	721.77	3480.87	480	5.6481875

Continúa en la siguiente página

Tabla 2 – Continación de la página anterior

map	shuffle	reduce	total	nmap	tumap
2732.22	48.54	794.76	3575.52	480	5.692125
2678.47	48.28	792.52	3519.27	480	5.580145833
2792.5	50.96	801.76	3645.22	480	5.817708333
2936.32	49.35	752.51	3738.18	480	6.117333333
2929.99	48.17	702.39	3680.55	480	6.104145833
2870.55	49.8	742.73	3663.08	480	5.9803125
2854.65	52.19	833.26	3740.1	480	5.9471875
		Promedio	3620.508163		5.889356293
		Desviación	83.70201133		0.168709194

Tabla 3: Tiempo en segundos para 3 nodos en m1.medium

map	shuffle	reduce	total	nmap	tumap
2122.16	67.88	506.48	2696.52	320	6.63175
1960.52	64.29	531.71	2556.52	320	6.126625
2075.28	67.03	521.42	2663.73	320	6.48525
2018.99	68.5	567.98	2655.47	320	6.30934375
2005.96	64.09	466.87	2536.92	320	6.268625
2137.49	69.75	505.3	2712.54	320	6.67965625
2048.4	66.31	469.92	2584.63	320	6.40125
1967.75	63.89	513.45	2545.09	320	6.14921875
2098.56	69.11	535.99	2703.66	320	6.558
2104.94	67.69	501.06	2673.69	320	6.5779375
2070.3	69.42	535.21	2674.93	320	6.4696875
2024.33	65.76	458.62	2548.71	320	6.32603125
1991.22	65.28	463.13	2519.63	320	6.2225625

Continúa en la siguiente página

Tabla 3 – Continación de la página anterior

map	shuffle	reduce	total	nmap	tumap
1951.89	64.92	533.55	2550.36	320	6.09965625
2090.46	67.04	466.1	2623.6	320	6.5326875
2059.42	68.46	549.91	2677.79	320	6.4356875
1979.65	64.34	507.74	2551.73	320	6.18640625
2058.36	66.2	480.88	2605.44	320	6.432375
1969.26	65.42	554.96	2589.64	320	6.1539375
2008.98	65.31	533.8	2608.09	320	6.2780625
1968.25	66.09	528.66	2563	320	6.15078125
1965.52	65.06	535.07	2565.65	320	6.14225
1972.3	65.04	542.52	2579.86	320	6.1634375
2042.02	68.76	539.08	2649.86	320	6.3813125
2086.15	70.15	553.17	2709.47	320	6.51921875
2077.4	67.64	518.63	2663.67	320	6.491875
2026.35	66.42	468.41	2561.18	320	6.33234375
2140.7	68.79	478.92	2688.41	320	6.6896875
1979.98	66.16	504.94	2551.08	320	6.1874375
2005.2	65.4	459.43	2530.03	320	6.26625
2042.61	66.79	469.83	2579.23	320	6.38315625
2038.66	66.77	559.66	2665.09	320	6.3708125
2034.78	66.75	509.09	2610.62	320	6.3586875
2064.12	67.67	494.56	2626.35	320	6.450375
2013.72	66.66	557.4	2637.78	320	6.292875
1979.97	67.13	558	2605.1	320	6.18740625
2043.07	66.73	469.26	2579.06	320	6.38459375
2048.39	65.77	462.36	2576.52	320	6.40121875
1979.69	63.95	498.55	2542.19	320	6.18653125
2001.68	64.44	490.13	2556.25	320	6.25525
1986.37	66.62	550.9	2603.89	320	6.20740625

Continúa en la siguiente página

Tabla 3 – Continación de la página anterior

map	shuffle	reduce	total	nmap	tumap
2023.84	64.64	475.43	2563.91	320	6.3245
2110.46	68.15	459.57	2638.18	320	6.5951875
2114.07	68.73	502.15	2684.95	320	6.60646875
1918.37	64.51	530.76	2513.64	320	5.99490625
2054.44	68.4	537.56	2660.4	320	6.420125
2088.04	69.16	548.51	2705.71	320	6.525125
1963.42	64.27	515.43	2543.12	320	6.1356875
2024.57	67.51	537.26	2629.34	320	6.32678125
		Promedio	2609.433265		6.348090561
		Desviación	57.85253454		0.167005883

Tabla 4: Tiempo en segundos para 4 nodos en m1.medium

map	shuffle	reduce	total	nmap	tumap
1528.68	77.33	550.69	2156.7	240	6.3695
1523.03	74.79	517.91	2115.73	240	6.345958333
1487.26	71.48	483.22	2041.96	240	6.196916667
1515.6	73.31	476.98	2065.89	240	6.315
1517.54	76.07	567.67	2161.28	240	6.323083333
1571.48	75.12	492.25	2138.85	240	6.547833333
1547.06	75.21	494.33	2116.6	240	6.446083333
1524.84	74.41	496.7	2095.95	240	6.3535
1571.18	76.11	520.64	2167.93	240	6.546583333
1497.78	72.55	466.9	2037.23	240	6.24075
1535.03	76.17	541.38	2152.58	240	6.395958333
1473.25	73.09	525.02	2071.36	240	6.138541667

Continúa en la siguiente página

Tabla 4 – Continación de la página anterior

map	shuffle	reduce	total	nmap	tumap
1539.22	74.91	475.32	2089.45	240	6.413416667
1476.06	72.68	483.14	2031.88	240	6.15025
1549.47	75.81	507.31	2132.59	240	6.456125
1484.86	73.05	498.53	2056.44	240	6.186916667
1537.94	76.98	537.86	2152.78	240	6.408083333
1557.21	77.35	550.9	2185.46	240	6.488375
1484.66	71.75	488.5	2044.91	240	6.186083333
1536.32	76.44	534.94	2147.7	240	6.401333333
1548.47	78.09	565.01	2191.57	240	6.451958333
1524.97	74.24	482.97	2082.18	240	6.354041667
1498.63	75.51	540.98	2115.12	240	6.244291667
1587.28	77.06	482.69	2147.03	240	6.613666667
1463.84	73.6	539.86	2077.3	240	6.099333333
1513.72	76.06	533.03	2122.81	240	6.307166667
1497.25	73.38	516.85	2087.48	240	6.238541667
1551.13	77.23	520.64	2149	240	6.463041667
1495.44	75.43	540.32	2111.19	240	6.231
1515.07	77.26	561.01	2153.34	240	6.312791667
1598.56	77.33	498.48	2174.37	240	6.660666667
1547.16	76.76	527.08	2151	240	6.4465
1486.67	74.87	526.39	2087.93	240	6.194458333
1597.49	76.98	504.56	2179.03	240	6.656208333
1514.72	75.92	561.77	2152.41	240	6.311333333
1540.94	75.07	508.86	2124.87	240	6.420583333
1504.76	75.22	533.89	2113.87	240	6.269833333
1565.18	77.25	553.64	2196.07	240	6.521583333
1510.66	77.38	568.46	2156.5	240	6.294416667
1547.16	78.38	558.75	2184.29	240	6.4465

Continúa en la siguiente página

Tabla 4 – Continación de la página anterior

map	shuffle	reduce	total	nmap	tumap
1459.54	72.99	527.74	2060.27	240	6.081416667
1531.33	76.15	512.05	2119.53	240	6.380541667
1604.64	78.08	508.61	2191.33	240	6.686
1492.42	73.69	520.72	2086.83	240	6.218416667
1533.4	76.1	504.64	2114.14	240	6.389166667
1518.89	73.42	475	2067.31	240	6.328708333
1569.4	75.72	508.23	2153.35	240	6.539166667
1554.57	77.21	528.73	2160.51	240	6.477375
1463.46	73.93	549.89	2087.28	240	6.09775
		Promedio	2121.656735		6.360137755
		Desviación	45.07276645		0.149577388

.1.2 MEDICIONES PARA M4.LARGE

Tabla 5: Tiempo en segundos para 1 nodo en m4.large

map	shuffle	reduce	total	nmap	tumap
3003.1	21.38	723.14	3747.62	960	3.128229167
2994.22	22.4	786.36	3802.98	960	3.118979167
3028.24	20.34	631.83	3680.41	960	3.154416667
3078	20.32	682.1	3780.42	960	3.20625
2970.49	19.68	775.92	3766.09	960	3.094260417
2976.73	21.92	717.64	3716.29	960	3.100760417
2973.01	21.51	712.71	3707.23	960	3.096885417
2840.45	20.75	706.52	3567.72	960	2.958802083
2826.55	20.88	735.02	3582.45	960	2.944322917

Continúa en la siguiente página

Tabla 5 – Continación de la página anterior

map	shuffle	reduce	total	nmap	tumap
3040.37	19.83	754.25	3814.45	960	3.167052083
3124.87	20.53	666.32	3811.72	960	3.255072917
2881.48	19.88	673.65	3575.01	960	3.001541667
3023.9	18.88	625.72	3668.5	960	3.149895833
2957.05	21.44	684.8	3663.29	960	3.080260417
2836.72	18.24	740.52	3595.48	960	2.954916667
2787.5	19.43	728.53	3535.46	960	2.903645833
2938.17	21.61	706.27	3666.05	960	3.06059375
2929.69	20.98	622.63	3573.3	960	3.051760417
2906.29	20.97	673.85	3601.11	960	3.027385417
2893.88	17.76	631.27	3542.91	960	3.014458333
2897.89	20.91	682.14	3600.94	960	3.018635417
2995.45	21.39	671.19	3688.03	960	3.120260417
2969.21	21.38	699.63	3690.22	960	3.092927083
3039.99	19.49	701.98	3761.46	960	3.16665625
2988.81	22.16	757.63	3768.6	960	3.11334375
3079.16	21.78	631.76	3732.7	960	3.207458333
2939.96	19.03	721.82	3680.81	960	3.062458333
2767.87	19.98	729.7	3517.55	960	2.883197917
3091.25	19.93	671.04	3782.22	960	3.220052083
2946.21	20.25	705.76	3672.22	960	3.06896875
2925.51	19.24	675.96	3620.71	960	3.04740625
3008.41	21.97	737.44	3767.82	960	3.133760417
2789.98	18.91	702.36	3511.25	960	2.906229167
2885.65	19.01	743.08	3647.74	960	3.005885417
2941.36	19.03	747.59	3707.98	960	3.063916667
2884.15	21.85	750.41	3656.41	960	3.004322917
2866.02	20.5	685.09	3571.61	960	2.9854375

Continúa en la siguiente página

Tabla 5 – Continación de la página anterior

map	shuffle	reduce	total	nmap	tumap
2840.74	21.47	716.76	3578.97	960	2.959104167
3064.34	21.15	661.14	3746.63	960	3.192020833
2874.88	20.25	645.41	3540.54	960	2.994666667
2979.84	21.63	702.23	3703.7	960	3.104
3015.99	21.62	749.88	3787.49	960	3.14165625
2861.12	21.64	737.45	3620.21	960	2.980333333
3063.38	19.66	638.33	3721.37	960	3.191020833
3003.17	21.36	708.36	3732.89	960	3.128302083
3021.53	19.73	686.31	3727.57	960	3.147427083
2820.43	19.27	690.35	3530.05	960	2.937947917
3047.33	19.77	746.21	3813.31	960	3.174302083
3124.93	20.93	649.05	3794.91	960	3.255135417
		Promedio	3674.987755		3.077067815
		Desviación	89.90588265		0.094182152

Tabla 6: Tiempo en segundos para 2 nodos en m4.large

map	shuffle	reduce	total	nmap	tumap
1637.91	31.96	380.97	2050.84	480	3.4123125
1525.69	29.65	326.54	1881.88	480	3.178520833
1467.54	27.88	348.44	1843.86	480	3.057375
1531.64	31.61	375.6	1938.85	480	3.190916667
1587.66	30.65	389.42	2007.73	480	3.307625
1514.15	30.45	386.43	1931.03	480	3.154479167
1488.27	29.18	332.76	1850.21	480	3.1005625
1645.4	30.92	331.9	2008.22	480	3.427916667

Continúa en la siguiente página

Tabla 6 – Continación de la página anterior

map	shuffle	reduce	total	nmap	tumap
1469.67	29.53	379.43	1878.63	480	3.0618125
1431.75	27.57	345.18	1804.5	480	2.9828125
1546.13	29.03	358.68	1933.84	480	3.221104167
1497.69	29.82	389.93	1917.44	480	3.1201875
1609.49	30.56	343.73	1983.78	480	3.353104167
1540.65	29.94	396.52	1967.11	480	3.2096875
1517.71	27.82	307.44	1852.97	480	3.161895833
1554.25	31.86	372.63	1958.74	480	3.238020833
1502.95	28.24	343.82	1875.01	480	3.131145833
1556.03	30.99	396.94	1983.96	480	3.241729167
1510.28	30.59	340.63	1881.5	480	3.146416667
1409.16	27.77	368.12	1805.05	480	2.93575
1516.83	28.87	337.32	1883.02	480	3.1600625
1644.72	31.4	356.29	2032.41	480	3.4265
1498.24	27.82	314.91	1840.97	480	3.121333333
1655.34	31.2	340.82	2027.36	480	3.448625
1482.99	28.58	382.48	1894.05	480	3.0895625
1617.63	31.66	407.87	2057.16	480	3.3700625
1523.42	29.09	343.44	1895.95	480	3.173791667
1449.91	30.08	378.29	1858.28	480	3.020645833
1537.97	28.87	350.91	1917.75	480	3.204104167
1556.85	29.54	358.29	1944.68	480	3.2434375
1617.38	31.28	336.71	1985.37	480	3.369541667
1635.12	33.25	371.81	2040.18	480	3.4065
1635.87	32.47	359.33	2027.67	480	3.4080625
1524.23	28.66	343.33	1896.22	480	3.175479167
1426.76	27.29	343.46	1797.51	480	2.972416667
1528.7	29.95	335.95	1894.6	480	3.184791667

Continúa en la siguiente página

Tabla 6 – Continación de la página anterior

map	shuffle	reduce	total	nmap	tumap
1644.18	31.92	371.55	2047.65	480	3.425375
1532.93	29.9	316.52	1879.35	480	3.193604167
1460.74	29.62	315.32	1805.68	480	3.043208333
1421.01	29.6	347.37	1797.98	480	2.9604375
1599.48	30.8	325.27	1955.55	480	3.33225
1538.15	32.07	382.99	1953.21	480	3.204479167
1639.45	33.23	386.58	2059.26	480	3.415520833
1637	33.63	368.65	2039.28	480	3.410416667
1464.34	29.76	382.35	1876.45	480	3.050708333
1615.22	30.83	340.18	1986.23	480	3.365041667
1497.51	29.16	354.53	1881.2	480	3.1198125
1610.27	32.12	352.16	1994.55	480	3.354729167
1526	30.5	327.67	1884.17	480	3.179166667
		Promedio	1928.752857		3.213531463
		Desviación	76.81066991		0.142616192

Tabla 7: Tiempo en segundos para 3 nodos en m4.large

map	shuffle	reduce	total	nmap	tumap
1197.63	43.03	275.59	1516.25	320	3.74259375
1264.91	44.98	289.86	1599.75	320	3.95284375
1295.86	46.63	272.48	1614.97	320	4.0495625
1229.22	44.36	262.77	1536.35	320	3.8413125
1121.1	41.19	276.86	1439.15	320	3.5034375
1249.56	43.57	246.23	1539.36	320	3.904875
1236.37	43.92	244.93	1525.22	320	3.86365625

Continúa en la siguiente página

Tabla 7 – Continación de la página anterior

map	shuffle	reduce	total	nmap	tumap
1179.03	43.07	287.35	1509.45	320	3.68446875
1263.27	44.28	258.35	1565.9	320	3.94771875
1221.66	43.43	248.52	1513.61	320	3.8176875
1245.18	44.39	263.47	1553.04	320	3.8911875
1222.33	43.22	263.64	1529.19	320	3.81978125
1157.8	41.96	266.15	1465.91	320	3.618125
1187	43.4	286.59	1516.99	320	3.709375
1222.36	42.88	246	1511.24	320	3.819875
1233.52	42.93	242.58	1519.03	320	3.85475
1137.08	39.83	226.5	1403.41	320	3.553375
1186.05	43.4	299.24	1528.69	320	3.70640625
1126.22	39.69	240.6	1406.51	320	3.5194375
1252.28	46.5	309.87	1608.65	320	3.913375
1291.44	45.47	250.39	1587.3	320	4.03575
1276.24	45.02	266.79	1588.05	320	3.98825
1155.3	42.02	280.43	1477.75	320	3.6103125
1246.06	43.84	274.79	1564.69	320	3.8939375
1303.19	45.16	257.84	1606.19	320	4.07246875
1187.74	43.01	260.58	1491.33	320	3.7116875
1294.96	45.43	252.83	1593.22	320	4.04675
1143.7	41.46	254.82	1439.98	320	3.5740625
1252.14	45.2	270.74	1568.08	320	3.9129375
1173.38	40.56	233.25	1447.19	320	3.6668125
1188.37	42.7	291.37	1522.44	320	3.71365625
1250.97	44.01	251.75	1546.73	320	3.90928125
1248.71	44.21	270.84	1563.76	320	3.90221875
1263.74	45.51	301.69	1610.94	320	3.9491875
1293.13	45.83	257.56	1596.52	320	4.04103125

Continúa en la siguiente página

Tabla 7 – Continación de la página anterior

map	shuffle	reduce	total	nmap	tumap
1196.62	43.24	267.26	1507.12	320	3.7394375
1180.29	43.23	282.75	1506.27	320	3.68840625
1197.94	41.99	247.1	1487.03	320	3.7435625
1193.25	43.96	287.29	1524.5	320	3.72890625
1158.76	41.37	251.99	1452.12	320	3.621125
1230.86	45.26	295.35	1571.47	320	3.8464375
1245.63	43.99	261.92	1551.54	320	3.89259375
1233.45	43.69	254.56	1531.7	320	3.85453125
1231.73	43.02	253.05	1527.8	320	3.84915625
1186.59	41.92	262.22	1490.73	320	3.70809375
1237.46	44.51	276.87	1558.84	320	3.8670625
1240.02	43.94	243.29	1527.25	320	3.8750625
1256.73	45.42	294.75	1596.9	320	3.92728125
1250.64	45	280.63	1576.27	320	3.90825
		Promedio	1530.946531		3.816165179
		Desviación	52.74564478		0.144301662

Tabla 8: Tiempo en segundos para 4 nodos en m4.large

map	shuffle	reduce	total	nmap	tumap
1002.44	50.86	197.29	1250.59	240	4.176833333
1027.5	52.66	204.74	1284.9	240	4.28125
1040.27	53.84	222.16	1316.27	240	4.334458333
941.95	49.12	230.97	1222.04	240	3.924791667
1004.69	51.27	208.16	1264.12	240	4.186208333
1043.21	52.74	219.24	1315.19	240	4.346708333

Continúa en la siguiente página

Tabla 8 – Continación de la página anterior

map	shuffle	reduce	total	nmap	tumap
960.27	49.28	219.12	1228.67	240	4.001125
968.07	48.35	182.04	1198.46	240	4.033625
935.13	47.34	200.39	1182.86	240	3.896375
1025.04	52.07	208.11	1285.22	240	4.271
990.03	52.07	232.12	1274.22	240	4.125125
938.29	48.88	228.69	1215.86	240	3.909541667
876.05	46.34	213.21	1135.6	240	3.650208333
1014.58	51.81	220.11	1286.5	240	4.227416667
937.88	49.3	222.99	1210.17	240	3.907833333
1020.26	51.98	226.64	1298.88	240	4.251083333
890.11	46.69	217.81	1154.61	240	3.708791667
1002.04	51.33	201.68	1255.05	240	4.175166667
896.5	46.16	191.7	1134.36	240	3.735416667
928.36	48.23	203.36	1179.95	240	3.868166667
955.48	49.85	223.32	1228.65	240	3.981166667
1022.18	53.63	246.41	1322.22	240	4.259083333
927.6	47.1	178.71	1153.41	240	3.865
903.21	45.81	181.44	1130.46	240	3.763375
1068.83	53.39	205.12	1327.34	240	4.453458333
999.64	50.9	221.19	1271.73	240	4.165166667
964.17	50.22	232.64	1247.03	240	4.017375
914.35	47.16	193	1154.51	240	3.809791667
935.5	47.18	195.04	1177.72	240	3.897916667
1027.95	53.86	240.37	1322.18	240	4.283125
1017.92	53.01	232.53	1303.46	240	4.241333333
957.93	48.25	197.09	1203.27	240	3.991375
1018.98	53.68	242.33	1314.99	240	4.24575
1027.33	52.49	219.41	1299.23	240	4.280541667

Continúa en la siguiente página

Tabla 8 – Continación de la página anterior

map	shuffle	reduce	total	nmap	tumap
988.89	49.87	182.49	1221.25	240	4.120375
975.55	49.39	204.86	1229.8	240	4.064791667
1020.58	52.2	205.95	1278.73	240	4.252416667
1037.08	53.13	232.51	1322.72	240	4.321166667
942.95	47.53	186.46	1176.94	240	3.928958333
972.83	49.95	206.74	1229.52	240	4.053458333
967.33	50.29	232.27	1249.89	240	4.030541667
982.58	49.24	181.69	1213.51	240	4.094083333
884.79	45.52	204.98	1135.29	240	3.686625
898.38	47.51	214.41	1160.3	240	3.74325
954.97	48.21	178.86	1182.04	240	3.979041667
905.62	46.34	181.36	1133.32	240	3.773416667
884.61	45.73	210.06	1140.4	240	3.685875
970.3	50.05	220.65	1241	240	4.042916667
1008.34	51.31	221.57	1281.22	240	4.201416667
		Promedio	1231.543878		4.045794218
		Desviación	61.36187412		0.206869026

.2 HISTORIAS DE USUARIO

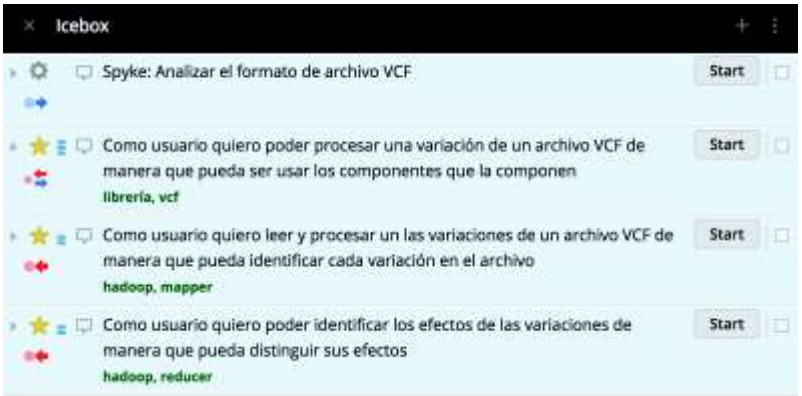

Figura 1: Icebox de la aplicación

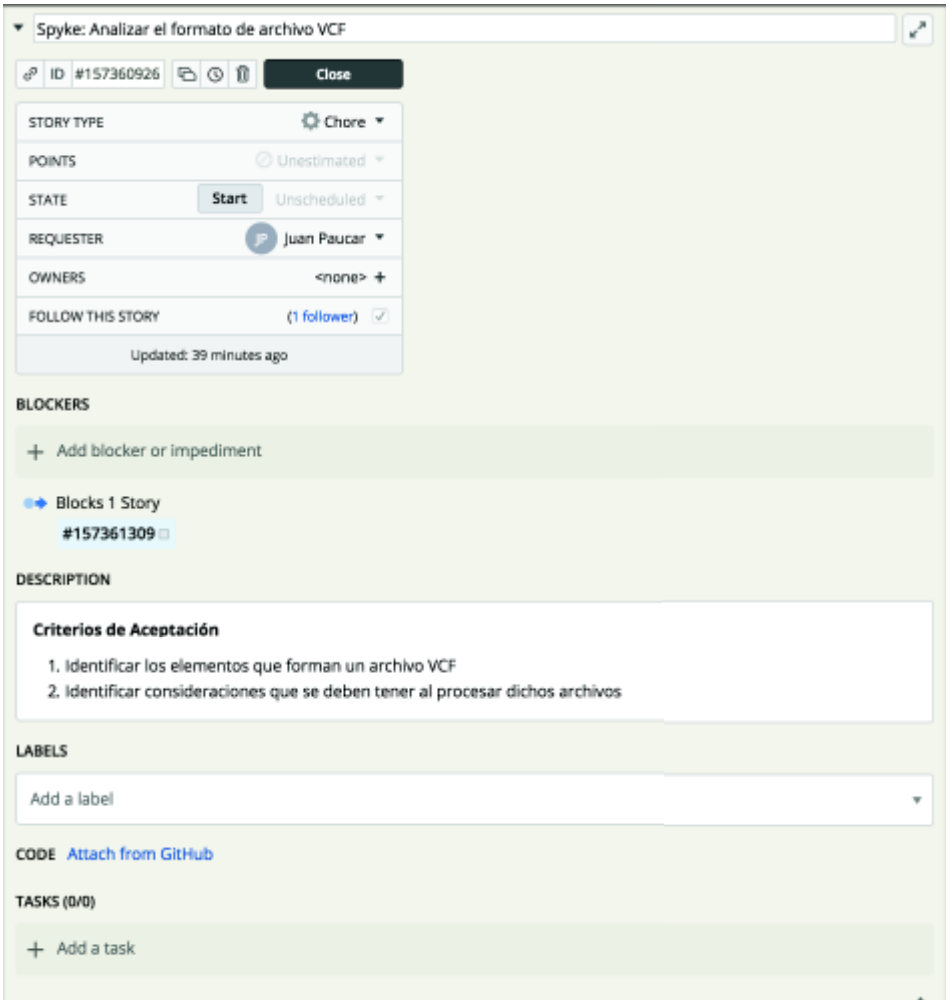

Figura 2: Historia de usuario 1

Como usuario quiero poder procesar una variación de un archivo VCF de manera que pueda ser usar los componentes que la componen

ID #157361309 Close

STORY TYPE	Feature
POINTS	3 Points
STATE	Start
REQUESTER	Juan Paucar
OWNERS	<none> +
FOLLOW THIS STORY	(1 follower)

Updated: 40 minutes ago

BLOCKERS

#157360926

+ Add blocker or impediment

Blocks 2 Stories

#157361530 #157362748

DESCRIPTION

Cómo usuario de la aplicación deseo poder procesar la información contenida dentro de archivos VCF. Para esto es necesario separar el procesamiento inicial de la información, de la aplicación en sí misma. Por lo que el procesamiento de la información debería hacerse en una librería

Criterios de Aceptación

1. El procesamiento las variaciones debe estar en una librería fuera del proyecto (para ser reusada después)
2. Se debe poder procesar las variaciones contenidas en un archivo VCF. (No es necesario procesar el resto de información en el encabezado)
3. Se debe tener pruebas unitarias y de regresión para validar que el procesamiento de una variación se da como se espera

LABELS

libreria × vcf ×

CODE Attach from GitHub

TASKS (0/2)

- [ ] Procesar datos de variaciones
- [ ] Procesar datos de pacientes

+ Add a task

Figura 3: Historia de usuario 2

Como usuario quiero leer y procesar un las variaciones de un archivo VCF de manera que pueda identificar cada variación en el archivo

ID #157361530 Close

STORY TYPE	Feature
POINTS	2 Points
STATE	Start
REQUESTER	Juan Paucar
OWNERS	<none>
FOLLOW THIS STORY	(1 follower)

Updated: 41 minutes ago

BLOCKERS

#157361309

Add blocker or impediment

DESCRIPTION

La primera parte del procesamiento es la lectura de un archivo VCF, el cual puede ser procesado usando la librería creada la historia previa. Este historia corresponde al *mapper* de la aplicación *MapReduce*

Consideraciones: Los archivos VCF, por lo general tienen un tamaño de varios GB. Es por esto que se debe ser cuidadoso de no mover todo el contenido del archivo a memoria. El archivo debería ser leído por partes o por líneas y usar una memoria constante.

Criterios de aceptación

1. La lectura del archivo debe ser a través del STDIN del sistema
2. Ignorar las líneas del encabezado del archivo (no son necesarias para el posterior análisis)
3. Procesar cada línea que contenga una variación
4. Escribir la salida con la llave valor al STDOUT. Siendo la llave la identificación de la variación y el valor la variación una vez procesada.

LABELS

hadoop × mapper ×

CODE Attach from GitHub

TASKS (0/2)

- [ ] Leer archivo VCF del STDIN
- [ ] Escribir al STDOUT la llave valor para el reducer

Add a task

Figura 4: Historia de usuario 3

Como usuario quiero poder identificar los efectos de las variaciones de manera que pueda distinguir sus efectos

ID #157362748 Close

STORY TYPE	Feature
POINTS	2 Points
STATE	Start
REQUESTER	Juan Paucar
OWNERS	<none> +
FOLLOW THIS STORY	(1 follower)

Updated: 41 minutes ago

BLOCKERS

#157361309

+ Add blocker or impediment

DESCRIPTION

Una vez identificadas las variaciones por el mapper, se puede buscar cuál es el efecto de cada una de las variaciones . No todas las variaciones se van a expresar en una persona. Así tampoco, no todas las variaciones tienen efectos adversos en la salud.

Esta historia corresponde al desarrollo de la parte de *reducer* de la aplicación.

Criterios de Aceptación

1. Leer el resultados llave-valor salida del *mapper* desde el STDIN del sistema.
2. Buscar efectos conocidos de la variación
3. Descartar aquellas variaciones que no tengan efectos negativos en la salud
4. Escribir aquellas variaciones relacionados con patologías en el STDOUT

LABELS

hadoop × reducer ×

CODE Attach from GitHub

TASKS (0/0)

+ Add a task

Figura 5: Historia de usuario 4