UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Análisis de factores de diagnóstico en informes radiológicos del proceso clínico diagnóstico de cáncer de mama mediante MapReduce

Departamento de Sistemas Informáticos y Computación

Master Universitario en Computación Paralela y Distribuida

Universitat Politècnica de València

Curso Académico 2015 - 2016

Autor

Luis Enrique Loor Maspons

Directores

Damián Segrelles Quilis

Germán Moltó Martínez

Valencia, septiembre de 2015

Abstract

Nowadays hospitals store a large amount of radiology reports in different Radiology Information Systems (RIS) from different modalities, such as ultrasound, MRI scans, CT scans, mammograms, X-rays, etc. These reports contain the data set observed by radiologists that can be of vital importance for the diagnosis of many diseases such as breast cancer.

This work designs, implements and analyzes different data analysis algorithms using the MapReduce paradigm to extract relevant factors to the diagnosis of breast cancer from the different diagnostic reports obtained in the process (mammography, ultrasound and MRI). These factors correspond to cross correlations between different modes and correlations with different grouping levels (understanding the level 'n' as the analysis of the correlation of 'n' parameters with diagnostic factor). These algorithms will perform distributed data processing on Hadoop clusters that are dynamically provisioned from public Cloud infrastructures.

Resumen

Hoy en día los hospitales almacenan en los diferentes Sistemas de Información Radiológicos (RIS) una gran cantidad de informes radiológicos de diferentes modalidades, como Ecografías, Resonancias magnéticas, Tomografías Axiales Computarizadas, Mamografías, Rayos X, etc. y con formatos variados. Dichos informes recogen el conjunto de datos observados por los radiólogos, además de aquella información obtenida al aplicar procesos sobre las propias imágenes (biomarcadores, etc.), que pueden ser de vital importancia para el diagnóstico de muchas enfermedades, como por ejemplo el cáncer de mama.

El presente trabajo va a diseñar, implementar y analizar diferentes algoritmos de análisis de datos utilizando el paradigma MapReduce para extraer a partir de los informes obtenidos en el proceso de diagnóstico de cáncer de mama (mamografías, ecografías y resonancias magnéticas) factores relevantes para el diagnóstico de cáncer de mama. Estos factores corresponderán a correlaciones transversales entre las distintas modalidades y correlaciones con diferentes niveles de agrupación (entendiendo el nivel ‘n’ como el análisis de la correlación de ‘n’ parámetros con el factor de diagnóstico). Dichos algoritmos procesarán de forma distribuida los datos en clusters Hadoop virtuales dinámicamente aprovisionados sobre infraestructuras Cloud públicas.

Agradecimientos

AL GOBIERNO NACIONAL DE LA REPÚBLICA DEL ECUADOR

Por apostar por mí y por muchos otros ecuatorianos que alrededor del mundo dejamos en alto el nombre de nuestro País, demostrando que SÍ SE PUEDE.

A MI ESPOSA E HIJA

Por estar siempre a mi lado, por apoyarme y alentarme en las decisiones que tomo… gracias por dejar todo y seguirme en mis ideales, que aunque a veces parezcan locuras, son siempre pensando en el bien de nuestro futuro.

A MIS DIRECTORES

Por guiarme con sus conocimientos durante la realización de este trabajo, por su paciencia y persistencia para siempre sacar lo mejor de mí y poder entregar un trabajo final de gran calidad.

A LOS PROFESORES DEL DSIC

A todos los profesores del Máster. Sin el conocimientos impartido por todos ellos, la realización de este trabajo no hubiese sido posible.

A AMAZON

Por la beca de educación otorgada a Germán Moltó y que ha permitido sufragar los gastos derivados del uso de recursos de cómputo, almacenamiento y red en la realización de este trabajo.

Tabla de Contenidos

Lista de Figuras

1. Introducción

Uno de los retos más importantes hoy en día en el mundo de la medicina, es la búsqueda de relaciones o patrones que ayuden en el diagnóstico de patologías.

A día de hoy, los informes médicos son uno de los contenedores más utilizados de información relativa a las enfermedades y los tratamientos aplicados a estas. Dichos informes, proceden de diversas y variadas fuentes. Sin embargo, muchos de ellos no cuentan con un tratamiento adecuado para generar conocimiento que permita la retroalimentación de información útil, y que sirva para identificar y medir la incidencia de determinados parámetros en el diagnóstico.

En este sentido, muchos centros médicos (hospitales, centros de atención primaria, etc.) cuentan con un gran volumen de datos, resultado de los estudios a los que son sometidos los pacientes en los procesos de diagnóstico, seguimiento o tratamiento de las enfermedades que presentan a lo largo de toda su vida. Toda esa información, por lo general, se encuentra almacenada en diferentes sistemas informáticos, con tipos de datos (texto, códigos, imágenes, videos, señales, etc.) y estructuras (informes sobre análisis clínicos, radiológicos, operaciones, etc.) variadas. En muchas ocasiones, toda esta información solo se guarda a nivel de registros históricos y no se utilizan para extraer la valiosa información que dichos datos pueden albergar escondidos, y menos aún si se considera la posibilidad de tratar dicha información como un conjunto global y transversal.

La posibilidad de poder contrastar esta información de forma global y transversal, podría ser de mucha importancia para el avance contra lucha de las muchas enfermedades que hoy en día atacan a nuestra sociedad, a través de la identificación y categorización de factores que proporcionen al clínico una herramienta para realizar diagnósticos más exactos, dejando menos margen a la subjetividad. En este sentido, la informática médica puede, a través del razonamiento inductivo, clasificar mediante diferentes técnicas [1], y crear modelos que permitan prospectivamente correlacionar parámetros entre informes. Por todo ello, la tendencia en la actualidad es la de diseñar modelos mixtos, que permitan combinar de forma transversal datos de fuentes y formatos diferentes, y que nos permitan saber con mayor exactitud la incidencia de determinados factores en los diagnósticos clínicos, como por ejemplo la probabilidad de padecer cáncer de mama [2] y su actuación médica posterior.

Por otro lado, el volumen de datos que se maneja y que se espera procesar, nos da una pauta para tener en cuenta que los tiempos de procesamiento pueden ser críticos a la hora de obtener los resultados y de ahí la necesidad de contar con sistemas escalables, capaces de adaptarse tanto al volumen de computación como al volumen de la información.

La forma tradicional de escalar nuestros sistemas, en ocasiones puede resultar demasiado complicado a la hora de escalarlos debido a la cantidad de actualizaciones y configuraciones que se necesitan hacer para añadir nuevo hardware y/o software. A parte de ser complicado, también puede llegar a ser costoso [3].

Hoy en día el Cloud Computing, o Computación en la nube, aparece como una alternativa a los sistemas tradicionales para ofrecer servicios de cómputo, almacenamiento y de red, pero con la diferencia en el acceso a ellos. El acceso a los servicios se hace a través de una red, que generalmente es Internet, y con la ventaja de que no se necesita tener ningún conocimiento de cómo gestionan los servicios.

Bajo este nuevo paradigma, se puede responder a un problema de forma flexible y dinámica, lo que permite atender sin inconvenientes grandes demandas de trabajo tanto de cómputo como de almacenamiento. La computación en la nube nos permite tener a disposición un gran número de servicios de una manera rápida y transparente. Entre los principales beneficios que nos ofrece el Cloud tenemos un aprovisionamiento de recursos en tiempo real, un alto grado de automatización y reducción de costes bajo el modelo de pago por uso [4].

Además, en el contexto de cómputo de unos grandes volúmenes de datos, no sólo se necesita una infraestructura escalable, sino también de aplicaciones igualmente escalables, que aprovechen los beneficios de la computación paralela. Bajo esta premisa aparece el modelo de programación MapReduce [5] que posibilita introducir soluciones tecnológicas basadas en computación distribuida a lo largo un conjunto de muchas máquinas o nodos.

En resumen, si hablamos de sistemas escalables que se adapten al cómputo y al volumen de información, y por otra parte tenemos volúmenes de información potencialmente considerables, donde los orígenes de la información son diversos y la velocidad para la obtención de los resultados es elevada, entonces podríamos ver este tipo de problema de búsqueda de correlaciones desde el punto de vista del BigData [6].

2. Motivación

El cáncer de mama es la principal causa de muerte por cáncer en mujeres, ya que anualmente se diagnostican más de un millón de casos a nivel mundial [7]. Este tipo de cáncer es con diferencia el tipo de cáncer más frecuente entre las mujeres, con aproximadamente 1,4 millones de nuevos casos de cáncer diagnosticados anualmente en todo el mundo (23% de todos los cánceres). Actualmente es el cáncer más común tanto en las regiones desarrolladas como las que se encuentran en fase de desarrollo.

El diagnóstico de cáncer de mama, es una de las enfermedades donde se genera un gran volumen de datos, generado por fuentes de información variadas (Mamografías, Ecografías, Resonancias Magnéticas, Exploración Clínica, Biopsias pre-Quirúrgicas, Biopsias post-Quirúrgicas, Tratamientos Neoadyuvantes, etc.) y en la que cada informe contiene una estructura e información diferente.

El Grupo de Grid y Computación de Altas Prestaciones (GRyCAP) de la Universidad Politécnica de Valencia (UPV) cuenta con una colaboración activa con el Hospital Universitario Doctor Peset y Hospital Universitario y Politécnico de la Fe (HPULF), en la que una de las áreas en las que se está trabajando es la recopilación estandarizada de información a través de informes estructurados siguiendo los estándares BI-RADS [8] (*Breast Imaging Reporting and Data System, BI-RADS*) y DICOM-SR [9] (*DICOM Structured Reporting*) relativos al diagnóstico, seguimiento y tratamiento del cáncer de mama [10].

Por ello, en la actualidad se dispone de un conjunto de informes estructurados relativos a episodios de diagnóstico de cáncer de mama en la que se han realizado las anotaciones de las imágenes adquiridas en estudios de mamografía, ecografía y resonancia magnética. La información contenida en estos informes está almacenada en documentos XML y en la que para

cada tipo de exploración tiene su propia estructura. Dicha información describe características como la forma de una lesión, el tipo, sus dimensiones, etc. siguiendo los términos que se describen en el estándar BI-RADS. Además, cada informe incluye una valoración o categoría definida también por BI-RADS de cada lesión anotada y que determina la probabilidad de padecer cáncer o no y las acciones a tomar por el médico. Por ejemplo, una mamografía puede tener una categoría de BI-RADS de 3 con los siguientes hallazgos: calcificaciones asociadas, redondas y de baja densidad. De acuerdo con los valores de BI-RADS, este estudio nos indicaría que el paciente tiene un resultado de probable benignidad pero que requiere control cada 6 meses.

La conclusión es que BI-RADS no considera en general datos que no sean procedentes del ámbito radiológico. Sin embargo, numerosos experimentos han estudiado la posible influencia de factores no radiológicos, como los provenientes de la exploración clínica, cambios en la evolución o antecedentes de riesgo. Un ejemplo claro sería el estudio de la presencia de sintomatología (fundamentalmente lesión palpable), que aunque no se ha demostrado que influya en la probabilidad de carcinoma en lesiones BI-RADS 3 (Probablemente benigna, con 2,25 % de probabilidad de cáncer) [11] [12] [13], sí parece influir dentro de la categoría BI-RADS 4 (probablemente maligna con sólo 2% de posibilidades de ser benigno). En [14] se demuestra a través de un estudio sobre 519 lesiones clasificadas como BIRADS 4, que el Valor Predictivo Positivo (VPP) en lesiones palpables fue del 54%, respecto al 16,8% en las no palpables, lo que justificaría la asignación de una subcategoría BI-RADS 4A (baja sospecha de malignidad 3% a 49%) o BI-RADS 4B (sospecha media de malignidad 50% a 89%) distinta en función de la presencia de sintomatología. Otro ejemplo claro es el mostrado en [15], en la que se demuestra que la incorporación de factores clínicos, epidemiológicos o de opinión del radiólogo basada en su propia experiencia permiten una mejor correlación con el grado de sospecha.

Las conclusiones BI-RADS tampoco valoran de forma transversal entre exploraciones radiológicas (por ejemplo cruzando información de una misma lesión proveniente de mamografía y ecografía), solo se basa en las anotaciones adquiridas en una misma exploración.

La motivación principal de este trabajo es la de desarrollar una plataforma que posibilite la realización de estudios capaces de analizar de forma transversal y global los datos asociados a los informes de diagnóstico de cáncer de mama tanto de ámbito radiológico como no radiológico, con el objeto de extraer correlaciones asociadas a las categorías de BI-RADS.

Los resultados que se pueden extraer de los estudios, pueden ser relevantes en el diagnóstico de cáncer de mama. Mediante estos estudios de identificación de patrones, se podría llegar a determinar la probabilidad de padecer cáncer con mayor exactitud. Dicho de otra forma, se podría conseguir una mejora en los Valores Predictivos Positivos (VPPs) y Negativos (VPNs) de las categorías definidas en BI-RADS y redundaría en una menor morbilidad, al evitar la realización de biopsias innecesarias o evitar retrasar su realización ante una sospecha.

Obviamente, la plataforma que se pretende desarrollar en este trabajo para encontrar la información asociada a los BI-RADS dentro de los informes, debe ser una plataforma informática que permita aprovechar las características de paralelización del problema planteado. Cada lesión es independiente de las demás, y por lo tanto podemos procesar varias lesiones de forma simultánea, en la que cada lesión tendrá asociada un conjunto de informes que la describen.

MapReduce parece un paradigma apropiado para abordar este tipo de problema, dado que es un modelo de programación que nos ofrece soporte a la computación paralela pero sin adentrarnos mucho en la misma, lo que es uno de los grandes inconvenientes de la computación distribuida. Este paradigma de programación está soportado por Apache Hadoop[16], que se ha convertido prácticamente en el estándar del mercado a la hora de hablar de BigData, utilizado por empresas como Yahoo, Facebook y eBay.

MapReduce se basa en el principio de divide y vencerás. Toma un conjunto grande de datos y los divide en pedazos pequeños e independientes entre sí, los cuales posteriormente se procesan en paralelo [17]. Enfocándonos en nuestro problema, tenemos una gran cantidad de informes estructurados, independientes entre sí y que por lo tanto pueden ser procesados de forma paralela.

Es por esto que este trabajo propone la búsqueda de relaciones entre características anotadas en los informes a través de algoritmos MapReduce.

3. Objetivos

El objetivo principal de este trabajo es diseñar una plataforma que permita realizar estudios que analicen bajo el paradigma MapReduce (implementado a través de Apache Hadoop), de forma transversal y global los datos asociados a los informes de diagnóstico de cáncer de mama en el ámbito radiológico, concretamente mamografías, ecografías y resonancias magnéticas, aunque también debe permitir la incorporación de otro tipo de informes de origen no radiológico (por ejemplo exploraciones clínicas, biopsias, etc.) para su análisis de una forma sencilla, con el objeto de extraer correlaciones asociadas a las categorías de BI-RADS.

Para la consecución de este objetivo principal se plantean los siguientes objetivos específicos:

- Diseñar e implementar una arquitectura escalable para resolver el problema planteado, que nos permita cubrir las necesidades de cómputo como las de almacenamiento. Además, debe de permitir flexibilidad a la hora de utilizar los recursos para no perder prestaciones.
- Validar la arquitectura a través de un conjunto de datos conformado por informes DICOM-SR de mamografías, ecografías y resonancias magnéticas, de forma que se extraiga el conocimiento relativo a las correlaciones existentes entre los las características BI-RADS y sus conclusiones.
- Visualizar el conocimiento generado en relación a los BI-RADS y su relación con las lesiones y características. La visualización debe ser entendible para permitir sacar conclusiones.

4. Estudio del Estado del Arte

A continuación se presenta el conjunto de tecnologías y estándares, que requieren ser clarificados para una mejor comprensión y justificación de la elección de las mismas para el diseño e implementación de la arquitectura que se plantea como objetivo en la sección 3.

4.1. Informes Estructurados DICOM-SR

En el este trabajo se han utilizado como fuentes de datos informes radiológicos relativos al diagnóstico de cáncer de mama que se presentan en ficheros XML y siguen el estándar DICOM-SR.

DICOM-SR se fundamenta sobre el estándar DICOM (Digital Imaging and Communications in Medicine), siendo este el estándar utilizado en la industria de la imagen médica, con el fin de ayudar al manejo, almacenamiento y transmisión de imágenes y datos asociados. El estándar define como ha de ser el formato de los ficheros contenedores de información, así como el protocolo de comunicación, permitiendo la integración de múltiples dispositivos dentro del mismo sistema de almacenamiento y comunicación [18][19].

El estándar DICOM se extendió para los informes que incorporan referencias a las imágenes, así como sus datos asociados. A dicho desarrollo se le denomina DICOM-SR (DICOM Structured Reporting) [9] [20]. Como cualquier otro objeto DICOM, DICOM-SR dispone de una cabecera y de contenido. En un informe estructurado DICOM-SR podemos encontrar información jerarquizada en distintos niveles, donde los elementos de información son de distintos tipos, como texto, valores numéricos, coordenadas espaciales o temporales y las referencias a las imágenes o formas de onda, tal como se puede observar en la Figura 1. Los elementos de información se encuentran conectados jerárquicamente en forma de árbol.

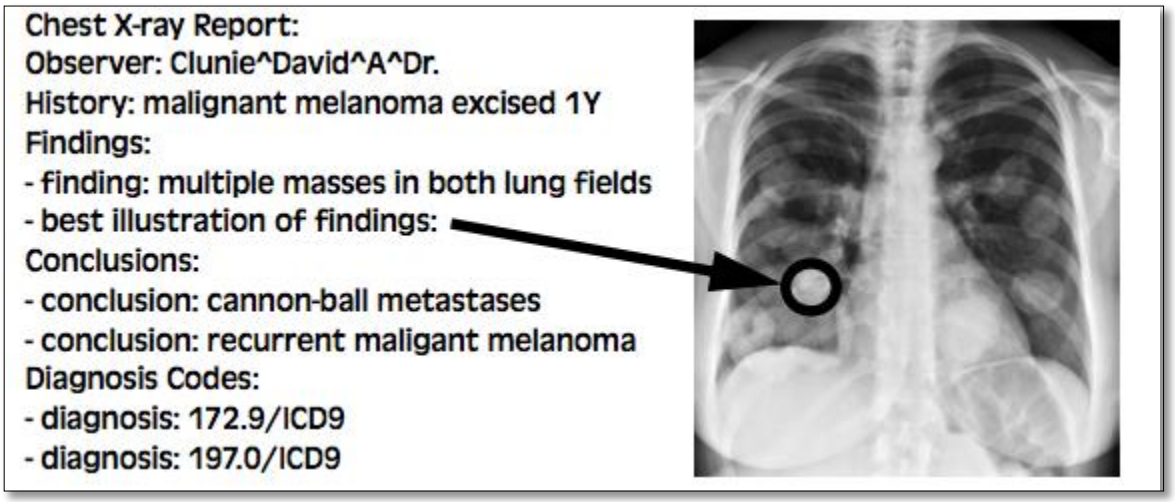

Figura 1: Ejemplo de DICOM-SR extraído de [4]

El estándar ha adoptado el lenguaje XML para codificar el documento estructurado. Esto posibilita que DICOM-SR puede ser transformado de forma muy fácil a otro formato de documento.

4.2. Infraestructuras HADOOP

Hadoop [16] es un framework que ofrece soporte escalable para aplicaciones distribuidas, esto debido a que permite trabajar clusters conformados por miles de nodos. Una de sus principales características es que está diseñado para trabajar con datos en el orden de los petabytes. Su funcionamiento está basado en MapReduce y Hadoop Distributed Filesystem. Con un cluster Hadoop abarcamos dos aspectos, la gestión de los trabajos MapReduce y el almacenamiento de datos.

Fue creado por Doug Cutting como apoyo al proyecto del motor de búsqueda Nutch y posteriormente renombrado como Hadoop al incorporarse la escalabilidad.

Yahoo fue uno de los principales beneficiarios de los servicios de Hadoop y en Junio de 2009 anunciaba que ofrecía su versión de producción de Hadoop disponible para el público y la comunidad opensource.

Al día de hoy la popularidad que ha alcanzado Hadoop ha hecho que en el mercado existan varias implementaciones alternativas como Hortonworks [22] o Cloudera [23] que han añadido nuevas funcionalidades y herramientas visuales que mejoran la experiencia administrativa del entorno y que en algún caso mejora el rendimiento. Además de poderse realizar instalaciones en centros de datos tradicionales, el auge del cloud en la actualidad ha hecho que las empresas de cloud público más importantes como Microsoft, Amazon y Google permitan desplegar infraestructuras virtuales con preinstalaciones de Hadoop que no necesitan de ninguna experiencia de configuración.

MapReduce [5] es un modelo de programación diseñado originalmente por Google y a través de este modelo se da soporte a la computación paralela sobre grandes cantidades de datos. Se basa en dos primitivas básicas, la función map() y la función reduce(), aunque internamente se ejecuten otras instrucciones. Para la función *map()* es necesario proveer pares tipo (clave / valor) de manera que produce una lista de valores para cada clave $Map(k_1, v_1) \rightarrow list(k_2, v_2)$. La función reduce se ejecuta para cada grupo k_2 con su lista de valores para producir un resultado agregado $Reduce(k_2, list\ (v_2)) -> list(v_3)$. La Figura 2 muestra el funcionamiento de un algoritmo MapReduce.

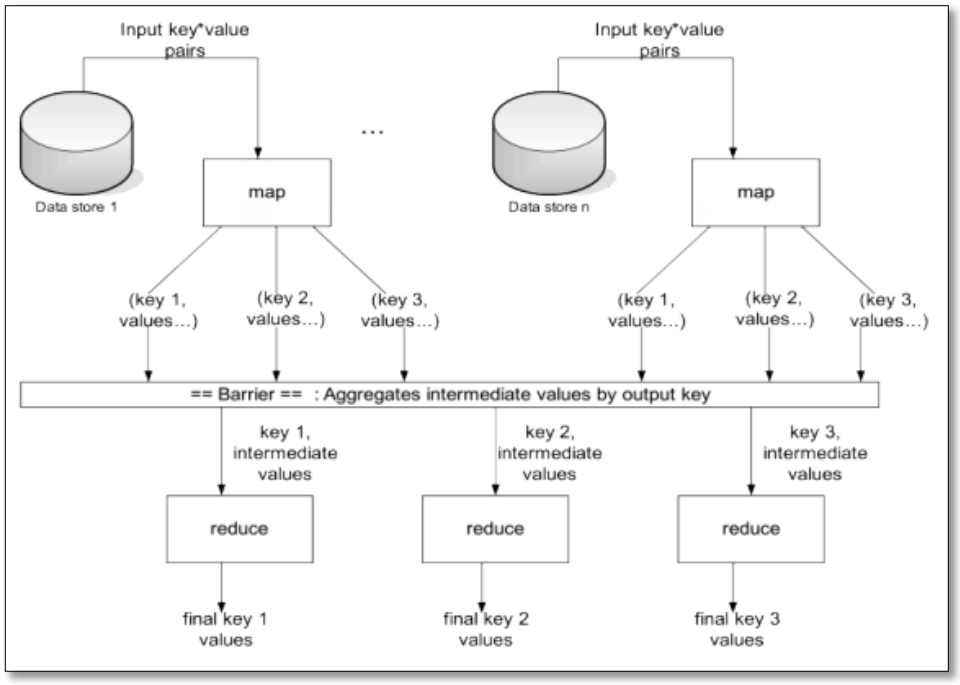

Figura 2: Funcionamiento de un algoritmo MapReduce

Se basa en el particionado de los datos, los cuales se procesan de forma paralela para posteriormente realizar un ordenado y combinado de los resultados parciales para obtener un resultado final (ver Figura 3).

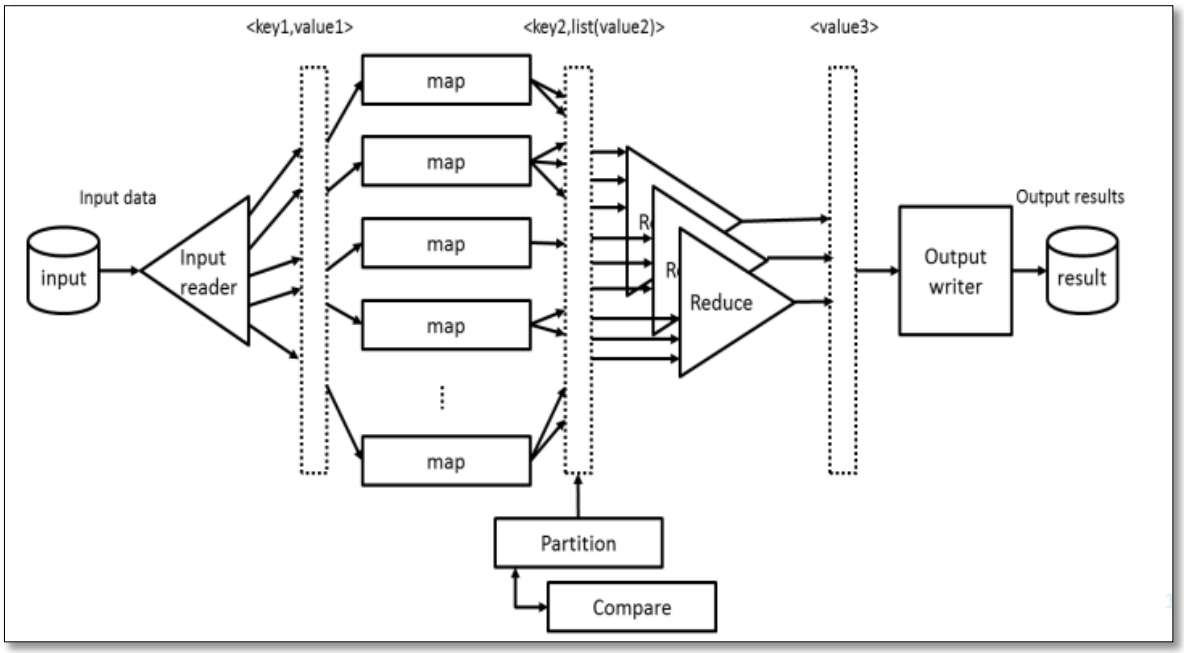

Figura 3: Ejemplo de procesamiento MapReduce

HDFS (Hadoop Distributed FileSystem) [24] es un sistema de ficheros distribuidos basado en Java diseñado para funcionar en hardware convencional o *commodity hardware*. En este sistema de ficheros la información se almacena en bloques de tamaño fijo (64 MB por

defecto) soportando tolerancia a fallos. Ofrece una gran escalabilidad llegando a soportar un cluster de más de 4000 servidores [25].

Su funcionamiento se basa en dos servicios principales, el NameNode y DataNode como se puede apreciar en la Figura4. El NameNode es el servicio que conoce dónde se encuentran los bloques de datos, a través de los metadatos que almacena. Con los metadatos sabe en qué servidor se encuentra cada bloque, cuantas réplicas tiene y donde están, etc. Además es quien rige el acceso de los clientes a los ficheros. Los DataNodes, por otra parte, almacenan los bloques de datos y realizan operaciones como creación de nuevos bloques, borrado y replicación de bloques, siempre bajo las órdenes del NameNode.

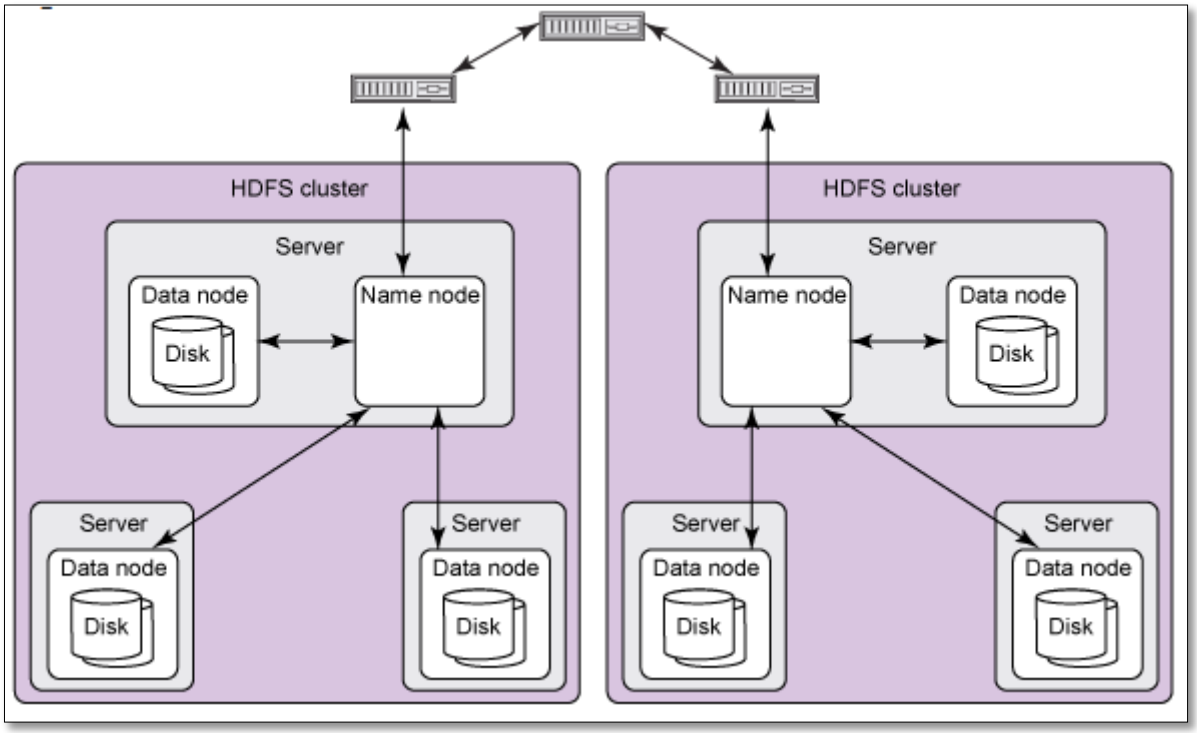

Figura 4: Arquitectura de HDFS

Como primera tentativa, en la implementación de los componentes de la arquitectura que se proponen como objetivo, podríamos realizar una instalación de un Cluster Hadoop desde cero, esta aproximación sería útil si nuestro objetivo fuera entender como configurar un Cluster, pero en nuestro caso como segunda opción podemos usar distribuciones que vienen con alguna implementación de MapReduce preinstalada y totalmente funcional.

Para la segunda opción nos podemos encontrar diferentes versiones, entre las más renombradas tenemos:

Distribuciones instalables por los usuarios

- Cloudera [23]. Es una empresa de software Americana que provee software basado en Apache Hadoop. Su principal producto es Cloudera´s Apache Hadoop

Distribution (CDH). Esta empresa se inició en el 2008 bajo la unión de 3 ingenieros de Google, Yahoo y Facebook.
- HortonWorks [22]. Fundado en 2011 por 24 ingenieros de Yahoo. Su principal producto es Hortonworks Data Platform (HDP).
- IBM BigInsights [26]. Plataforma Analítica para BigData ofrecida por IBM. Es 100% open-source e incluye características adicionales para el análisis de datos.

Servicios gestionados en la nube

- HDInsight de Azure [27]. Propuesta de Microsoft para Hadoop diseñado para la nube con lo cual asegura que se puede escalar en los datos bajo demanda.
- Amazon Elastic MapReduce (EMR) [28]. Propuesta de Amazon Web Services (AWS) para el procesamiento de grandes cantidades de datos. Integra el marco del trabajo de Hadoop con otros componentes del ecosistema de AWS como Amazon EC2 para el aprovisionamiento dinámico de máquinas y Amazon S3 para el almacenamiento.

Por cuestiones de disponibilidad de infraestructura utilizaremos Amazon EMR, que permite la configuración y el despliegue automatizado de clusters Hadoop sobre infraestructura virtual dinámicamente aprovisionada de Amazon EC2. Esto simplifica el proceso de creación de clusters Hadoop virtuales. Si bien es cierto que AWS es un proveedor de Cloud público de pago, se han sufragado los gastos mediante una beca de educación proporcionada por AWS. Esto ha permitido desplegar custers Hadoop sin incurrir en costes adicionales.

Además de la disponibilidad, existen múltiples razones por las cuales se puede elegir Amazon EMR para desplegar clusters Hadoop, entre ellas tenemos:

- Costos relativamente bajos: En relación a comprar el hardware, alquilar capacidad de cómputo de Amazon puede ser incluso más económico. De acuerdo a lo que indica la web oficial de AWS, se pueden lanzar un cluster de 10 nodos a tan solo 0.15 USD la hora [28].
- Facilidad de uso: Se puede desplegar un cluster completamente funcional con unos pocos pasos y en cuestión de minutos.
- Elasticidad: Permite el aprovisionamiento dinámico de instancias de manera que se puede dimensionar el tamaño del cluster Hadoop en función del tamaño de problema.

4.3. Almacenamiento en sistemas Hadoop

La obtención de los ficheros DICOM-SR en formato XML no es parte del presente trabajo, ya que damos por entendido que disponemos de ellos. Sin embargo debemos almacenarlos de una manera óptima dentro del sistema de ficheros HDFS del cluster Hadoop a utilizar en el proyecto.

Debemos tener en cuenta que nuestros ficheros XML corresponden a informes individuales de tamaño considerablemente pequeño, en el orden de los kilobytes. Si tratamos el problema

de la manera como está planteado, no es una buena idea usar Hadoop puesto que es menos eficiente cuando almacenamos un número grande de ficheros pequeños. Un fichero es considerado pequeño si es considerablemente más pequeño que el tamaño de bloque definido en HDFS que es de 64MB, aunque en otras distribuciones de Hadoop como las de IBM BigInsigths [26] y Hortonworks [22] llega a ser de 128MB y 256MB respectivamente. Cundo se quieren utilizar ficheros considerados como pequeños se enmarca dentro del llamado *el problema de los ficheros pequeños (the small files problem)* [34] [35] [36].

Este problema ocurre a dos niveles, tanto en el sistema de almacenamiento HDFS como en la gestión de los datos durante la ejecución del trabajo MapReduce.

En el primer caso, es importante destacar que HDFS no está diseñado para poder manejar muchos ficheros. El problema radica en el servicio *NameNode* encargado de guardar la ubicación de cada uno de los bloques dentro de un cluster hadoop. En la memoria de un *NameNode* se almacenan referencias a objetos que representan a ficheros y/o directorios dentro del sistema HDFS. Dado que cada representación ocupa alrededor de 150 bytes, almacenar 1 millón de ficheros pequeños requiere una cantidad de memoria física de alrededor de 0.25GB. En la siguiente tabla se muestra una estimación del tamaño en Gigabytes que necesitaría un *NameNode* a medida que crece el número de ficheros.

# ficheros(millones)	kilobytes	megabytes	gigabytes
1000000	292968,75	286,1022949	0,279396772
10000000	2929687,5	2861,022949	2,793967724
100000000	29296875	28610,22949	27,93967724

Tabla 1 - Estimación en GB del tamaño para la memoria de un NameNode

Como se puede observar a medida que aumenta el número de ficheros almacenado en HDFS, la memoria que utiliza el NameNode crece considerablemente, limitando la escalabilidad de dicho servicio.

Además del problema indicado anteriormente, existen otros inconvenientes en HDFS al almacenar muchos ficheros pequeños. Si tomamos el valor de 10 millones de ficheros vemos que aproximadamente debe tener 3GB para almacenar en memoria lo cual supone que en el eventual caso de un reinicio del servicio NameNode, éste deberá leer desde disco los metadatos de cada fichero para volver subirlos a memoria principal, esto teniendo en cuenta el valor de 3GB supondrá un retardo considerable a la hora de volver a levantar el servicio.

El segundo inconveniente viene asociado al rendimiento al ejecutar MapReduce y esto es fácil deducir dado que si tenemos muchos ficheros pequeños, entonces tendremos que tener mucho acceso a disco. En cualquier sistema uno de los principales problemas a resolver son los accesos físicos a disco puesto que los tiempos de acceso son más lentos que otros componentes como por ejemplo la memoria RAM.

Para nuestro problema, de antemano ya conocemos los tamaños de los ficheros XML y con esto podremos hacer una estimación cuando podríamos necesitar de almacenamiento. Teniendo en cuenta que el trabajo contempla ficheros XML que contienen los resultados de

tres informes, la siguiente tabla muestra el espacio de almacenamiento necesario en Gigabytes considerando 108 KB por fichero.

# XML	sistema ficheros local
50000	5,15
100000	10,30
200000	20,60
300000	30,90

Tabla 2. Tamaño estimado en GB para almacenamiento de informes

El sistema de almacenamiento propio de Hadoop es HDFS, sin embargo soporta otros tipos de sistema de almacenamiento externo como Amazon S3, Windows Azure Blob Storage, CassandraFS, CephFS, Google Cloud Storage Connector, MapR FileSystem. En el caso de almacenamiento externo, en algunos casos como Amazon S3 [30], existe un proceso de precarga de los ficheros al HDFS local, ya que así se puede acercar la computación a los datos.

Existen 3 técnicas de almacenamiento que nos ayudan a solventar el problema de la gestión de muchos ficheros pequeños.

Hadoop ARchives (HAR) [31]

Nos permite empaquetar ficheros y/o directorios en bloques HDFS de manera mucho más eficiente y que son válidas como entradas a trabajos MapReduce. Se componen de dos índices y partes de ficheros. El primer índice (master index) permite buscar una parte de un fichero y en qué parte se encuentra y el segundo (index) nos permite buscar el desplazamiento y la longitud. La figura 5 muestra cómo está estructurado un HAR file

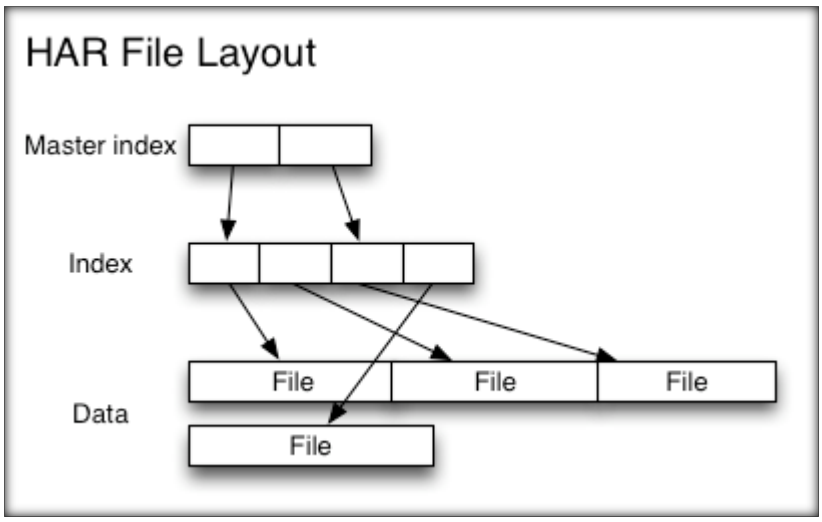

Figura 5: Estructura de un HAR file

Está solución es bastante adecuada para evitar el problema de los ficheros pequeños, si bien no está exenta de inconvenientes:

- Al momento de crear un fichero HAR se hace una copia de los ficheros originales a incluir en el .har, con lo cual es necesario espacio en disco temporal adicional. Además no soporta la compresión de los ficheros.
- Los ficheros .har son inmutables. Una vez creados no se pueden modificar.
- A pesar de ser una entrada válida para trabajos MapReduce, aún sigue siendo ineficiente a la hora de buscar las partes de los ficheros puesto que necesita buscar en dos índices.

SequenceFiles [32] [33]

Los SequenceFiles ofrecen la alternativa de almacenar de forma binaria pares clave / valor como se muestra en la Figura 6. Ofrecen compresión, una ventaja frente a los ficheros HAR, permitiendo ahorrar espacio. Su principal característica es que son particionables, es decir que se puede dividir y enviar cada trozo a diferentes mappers.

Figura 6: Estructura de un SequenceFile

Los niveles de compresión de los que dispone son:

- Sin compresión.
- Compresión a nivel de registro. Sólo los valores se comprimen
- Compresión a nivel de bloque. Tanto clave como valor se comprimen en bloques separados.

Empiezan con una cabecera (ver Figura 7) que contiene la información necesaria para poder leer el fichero. Básicamente contiene la versión, el tipo de clase de la clave, tipo de clase del valor, indicador si está comprimido o no, entre otros.

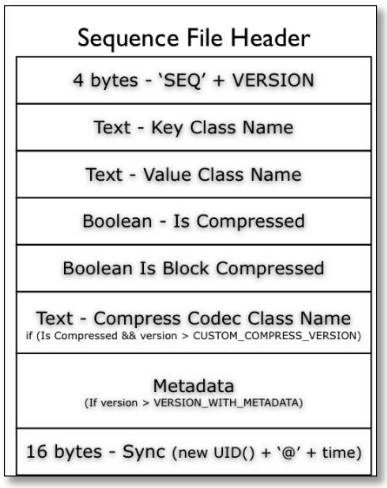

Figura 7: Cabecera de los SequenceFiles

En cuanto a la compresión, no existe mayor diferencia en cómo se representan las estructuras, entre con o sin compresión. La diferencia radica en que los registros que tiene compresión, el valor se encuentra comprimido como se aprecia en la Figura 8.

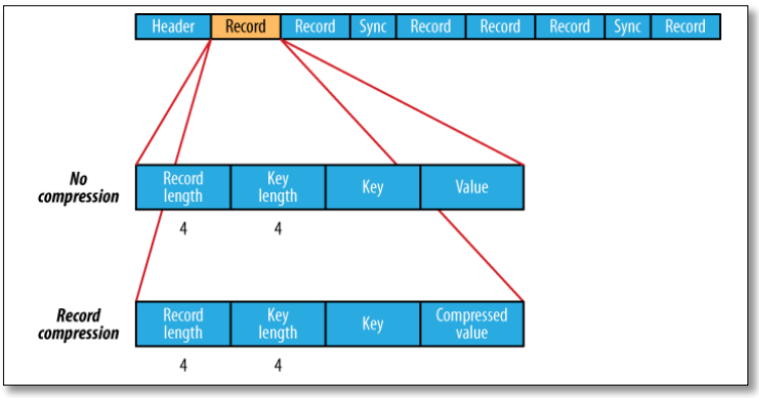

Figura 8: SequenceFiles, almacenamiento con y sin compresión

En la compresión a nivel de bloque existe mayor compresión puesto que se comprimen tanto las claves como los valores. La figura 9 muestra una estructura SequenceFile con compresión en bloque.

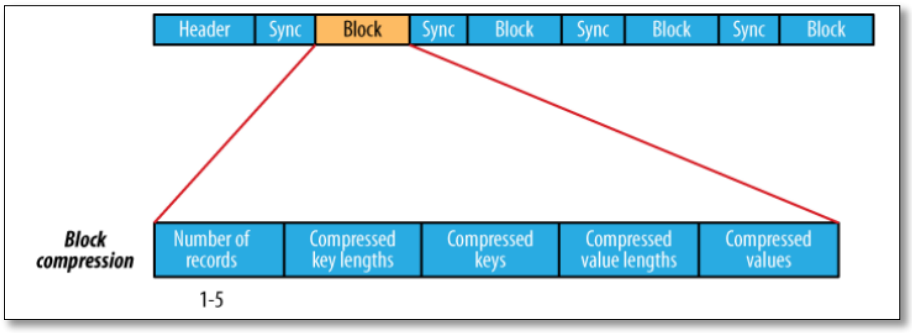

Figura 9: SequenceFile, compresión a nivel de bloque

Por las características que hemos presentado vemos que los SequenceFiles ofrecen algunas ventajas como la reducción del uso de espacio en disco, menor uso de ancho de banda y, lo más importante, es que son particionables.

MapFiles [31]

Es un directorio que contiene dos ficheros, el primero es el de los datos, que contiene las claves y los valores, y el segundo un pequeño índice que se compone con una parte de la clave (debido a que se está enteramente en memoria). La figura 10 nos muestra cómo está estructurado un MapFile.

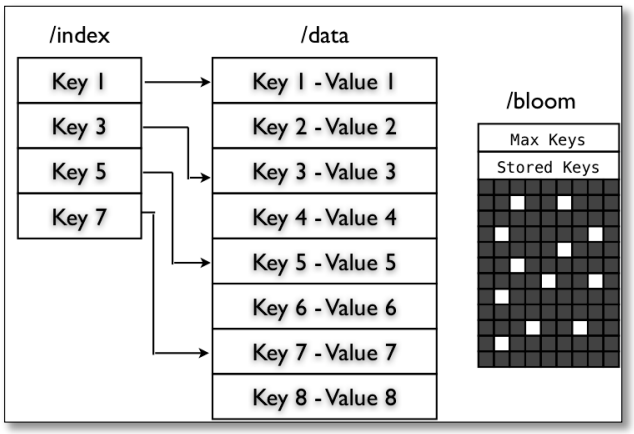

Figura 10: Estructura de un MapFile

Una vez que hemos analizado los beneficios e inconvenientes de cada uno de los formatos para almacenamiento que nos ofrece Hadoop y teniendo en cuenta las características de nuestro problema, nos inclinamos por utilizar los SequenceFiles puesto que no necesitamos los ficheros originales una vez comprimidos (har files) y tampoco es necesario ningún tipo de indexación (MapFiles) para acceder a los ficheros, ya que no importa el orden de acceso a los mismos.

4.4. Software de Visualización

Algunas de las posibilidades que nos podemos encontrar a la hora de presentar los resultados son las siguientes:

- R [37], un lenguaje de programación para análisis estadístico y la generación de gráficos. Este lenguaje opensource se derivó del lenguaje S y se distribuye bajo licencia GNU GPL. Fue creado por Robert Gentleman y Ross Ihaka del Departamento de Estadística de la Universidad de Auckland y es muy usado en la investigación siendo muy popular en el campo de la biomédica, bioinformática y las matemáticas financieras. Aparte de la instalación básica de R, se pueden añadir funcionalidades adicionales a través varios paquetes que se encuentran disponible en el repositorio de R. El amplio uso de R, se debe además a su capacidad de generar gráficos de gran calidad que permiten su publicación.
- Tableau [38], es un software que ofrece análisis de datos. Permite que los datos puedan residir en la nube o en las aplicaciones. Ayuda a identificar patrones, ver tendencias y descubrir información visual.
- GnuPlot [39], es una utilidad de línea de comandos disponible en varios sistemas operativos que permite crear gráficos. Inicialmente fue creado para visualizar funciones matemáticas y datos interactivos, sin embargo su soporte ha ido en aumento que ahora incluso es el motor de gráficos de herramientas terceras como Octave.

Dado el amplio uso de R en el ámbito científico y la complejidad de gráficos que nos permite generar vamos a usar esta herramienta para la visualización de resultados sin perjuicio de uso de las demás herramientas.

5. Arquitectura

5.1. Etapas del proceso

Para la identificación de los componentes que van a conformar la arquitectura de la plataforma propuesta como objetivo, se han identificado los pasos de todo el proceso de generación del conocimiento, que son las siguientes:

- ***Paso 1***, Extracción de los XML con los informes: Este paso inicial consiste en la recopilación de los ficheros XML estructurados que contienen los 3 tipos de informes (la mamografía, la ecografía y la resonancia magnética). Estos ficheros deben ser subidos a un repositorio centralizado. Este paso no forma parte del presente trabajo, se asume que los informes ya están generados y por lo tanto se puede hacer uso de ellos.
- ***Paso 2***, Compactación de los XML: Como vamos a usar Hadoop, tenemos que tener muy claro que Hadoop no es muy eficiente para procesar ficheros pequeños (nuestros XML lo son), dado que esta optimizado para procesar ficheros grandes, debido al problema presentado en el estado del arte en la sección 4.3 relativo al *problema de los*

ficheros pequeños. Como solución al problema, se utilizarán los SequenceFiles, tal y como se exponen en esta misma sección. En este paso crearemos un SequenceFile que contenga muchos XML. De esta manera pasaremos de tener muchos ficheros XML pequeños, a tener pocos ficheros SequenceFiles pero grandes. Para crear estos ficheros ejecutaremos un trabajo MapReduce que será el encargado de leerlos y empaquetarlos. Con esto ya tenemos preparados los datos para ser procesados de manera eficiente con Hadoop.

- ***Paso 3***, Despliegue del Cluster Hadoop: Vamos a desplegar un Cluster Hadoop con varios nodos de procesamiento (DataNodes). Además se deberán subir al repositorio los ficheros XML, los SequenceFiles y los JAR de Java con los algoritmos MapReduce implementados, que serán ejecutados en el Cluster Hadoop. Una vez que se han leído todos los ficheros de entrada, después de la ejecución del trabajo quedarán el/los ficheros de salida que contienen los datos necesarios para poder establecer la relación de los atributos y características.
- ***Paso 4,*** Presentación del conocimiento generado con R: A partir de la información resultante del procesamiento con Hadoop, utilizaremos R para crear un gráfico estadístico que permitirá visualizar el conocimiento oculto en los XML de los informes de cáncer de mama.

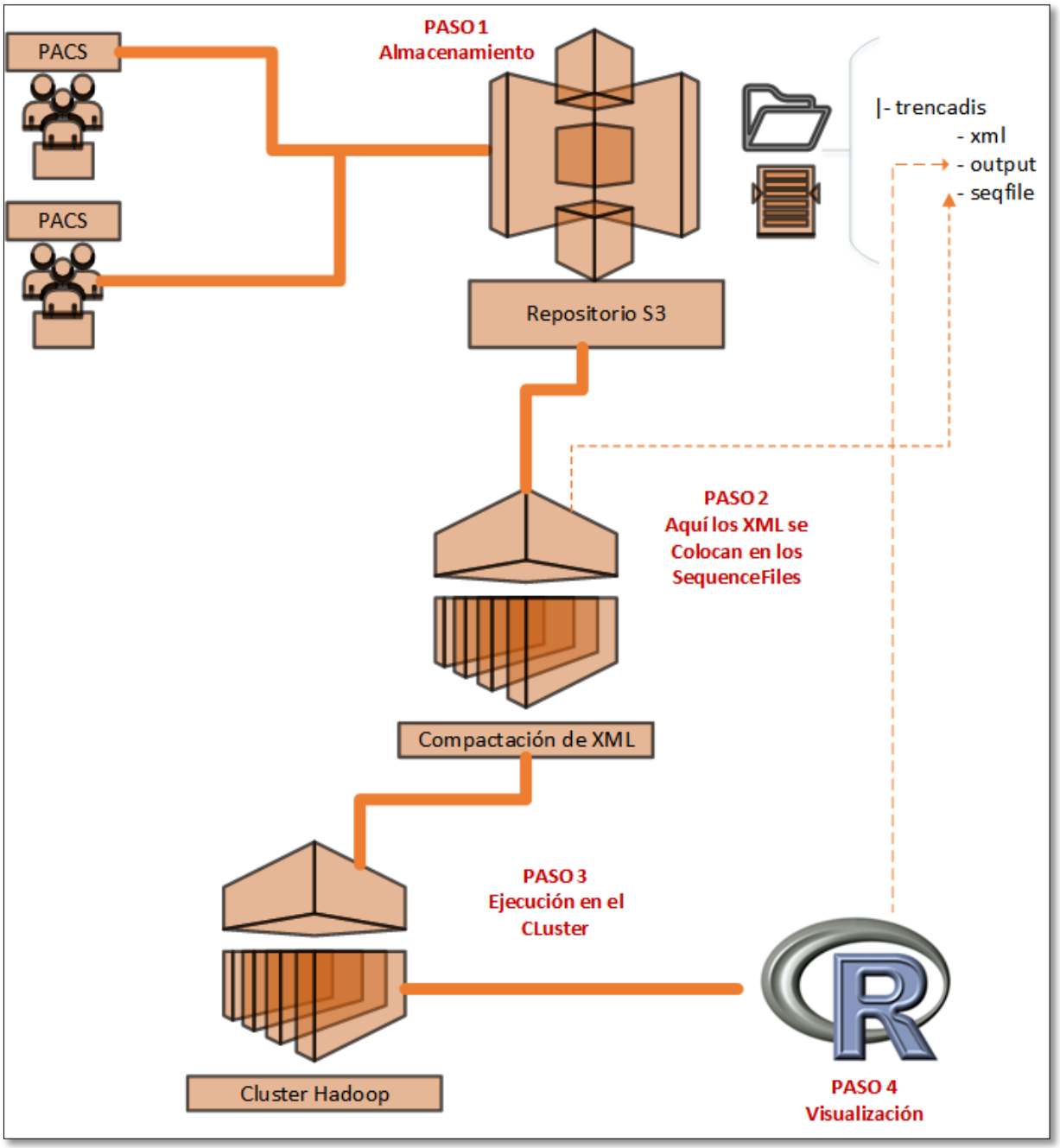

Figura 11: Diagrama pasos identificados en la aplicación.

5.2. Requerimientos de Arquitectura

Los requerimientos que se definen para la arquitectura son los siguientes:

- La arquitectura debe ser escalable, que se adapte a las tanto a las necesidades de cómputo como de almacenamiento. Tal y como se comenta en el estado del arte, en la sección 4.2, existen varias alternativas, desde adquirir nuestro propio hardware y construir nuestro sistema o utilizar proveedores de infraestructuras externos de los cuales hay varios en el mercado.

- La arquitectura debe disponer de un componente como repositorio central, en donde todos los centros médicos involucrados en los estudios de enfermedades puedan depositar los informes que dispongan. Necesitamos un repositorio de ficheros que nos ofrezca tolerancia a fallos y nos permita recuperarnos a ellos de una manera transparente. Además, debe ser un contenedor que nos ofrezca confiabilidad respecto a los datos que almacena, inclusive en los casos de fallo. Y sobre todo que pueda escalar cuando aumente el volumen de información.
- La arquitectura requiere una infraestructura elástica que permita flexibilidad a la hora de utilizar los recursos para la computación de información. Se debe poder manejar el crecimiento del trabajo sin perder prestaciones y que esté preparado para adaptarse a las circunstancias cambiantes.

Dadas las necesidades de nuestra arquitectura, vamos a construirla a través de la computación en la nube, utilizando una Infraestructura como Servicio (IaaS) en donde podemos utilizar tanto el almacenamiento como el despliegue dinámico de máquinas virtuales. Tal y como se ha comentado en el estado del arte en la sección 4.3, en este trabajo se va a utilizar Amazon Web Services como proveedor Cloud, dado que ofrece su servicio de almacenamiento llamado Amazon S3 con características de escalabilidad y alta disponibilidad. Por otro lado también se pueden realizar despliegue de máquinas con software preinstalado, tal y como se explica en la sección 4.2 del estado del arte, pudiendo utilizar el servicio Amazon Elastic MapReduce para el despliegue de clusters Hadoop.

Identificados los requerimientos, se ha optado por la definición de una arquitectura, la cual se presenta en la Figura 12 y que se detalla a continuación.

- Como repositorio centralizado de la información vamos a utilizar S3 porque nos ofrece almacenamiento persistente con alta disponibilidad y escalabilidad.
- Para el procesamiento necesitamos un cluster Hadoop de varios nodos el cual lo desplegaremos con EMR dado que es fácil de configurar e iniciar, además que una vez listo está totalmente funcional. Con esto la arquitectura quedaría de la siguiente manera.

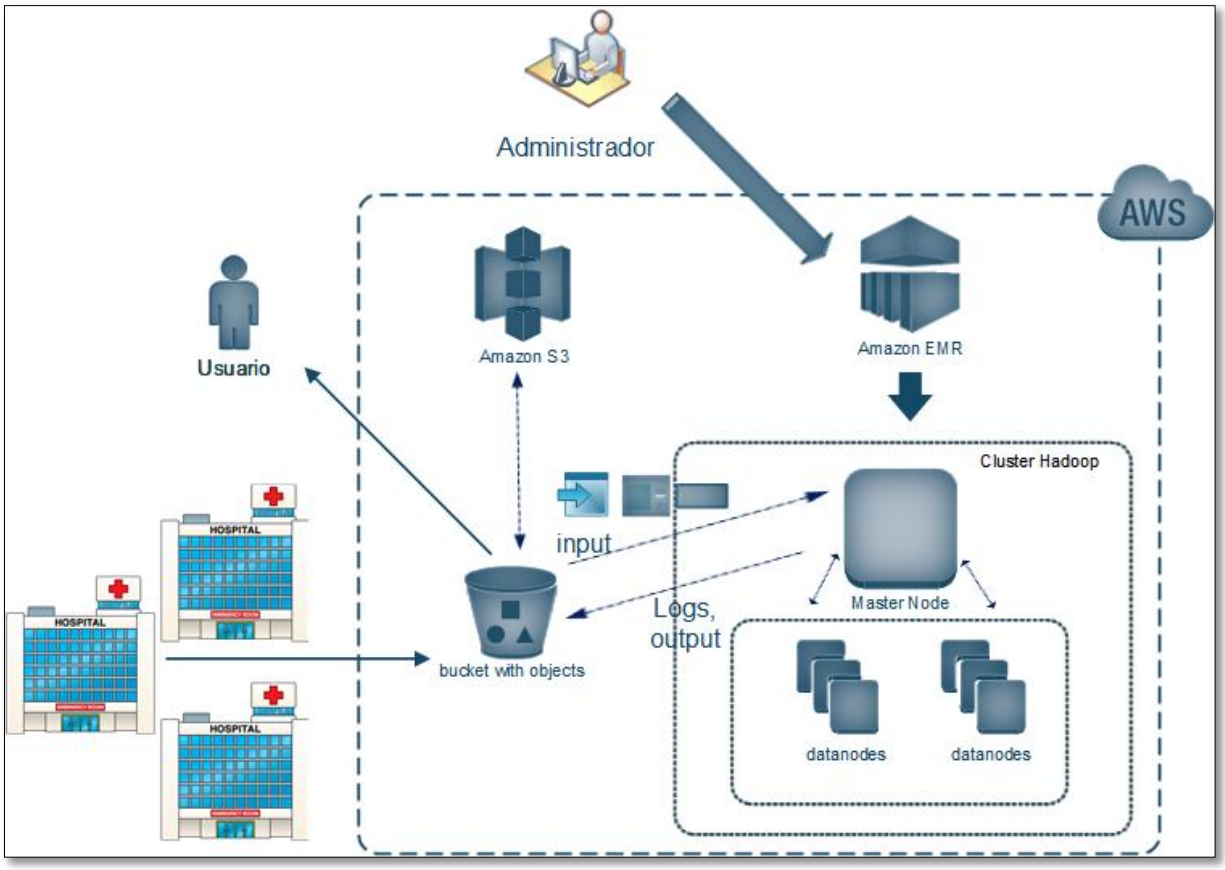

Figura 12: Arquitectura de la aplicación

6. Implementación de caso de uso

Tal y como se indicó en la sección 3 relativa a los Objetivos, una vez que tenemos definida la arquitectura, como segundo objetivo se propone validar la arquitectura a través de la implementación del caso de estudio relativo a la extracción de correlaciones de los parámetros BI-RADS con sus conclusiones.

La información de los estudios de cáncer de mama debe ser subida a nuestro repositorio de S3 en la ruta /trencadis/xml con la finalidad de que podamos compactarlos en un SequenceFile. Cuando los ficheros se encuentren disponibles procederemos a ejecutar un trabajo MapReduce que se encarga de recorrer el directorio /trencadis/xml, obtener cada fichero y colocarlo dentro de un SequenceFile los cuales se depositan en directorio diferente /trencadis/seqfile.

De igual manera se deberán subir todos los ficheros de las aplicaciones creados para la ejecución de los trabajos en un repositorio de S3, ya que desde aquí el cluster Hadoop tomará el código para empezar la ejecución de los trabajos. Los ficheros que están en S3 se copiarán automáticamente al HDFS del cluster al momento de lanzarse los trabajos.

Como siguiente paso, necesitamos especificar el desarrollo de los algoritmos que permiten resolver nuestro caso de uso de los BI-RADS en los informes de cáncer de mama.

Como se ha indicado en secciones anteriores, nuestra entrada de datos son ficheros XML como los mostrados en la Figura 13, que contienen los informes de cáncer de mama (resonancia, mamografía y ecografía). En MapReduce por defecto cada fichero de entrada en HDFS de parte (Split) según un delimitador que por defecto es el fin de línea. Para resolver nuestro problema, este enfoque no es viable puesto que en este caso necesitamos procesar cada fichero como un registro completo [8], debido a que no podemos tener un fichero XML fraccionado puesto que sería un XML mal formado.

La opción para resolver este inconveniente es la que ya describimos en la sección 4.3 relativo al *Almacenamiento en sistemas Hadoop,* cuya solución propuesta es la de usar los SequenceFiles, debido a que esto se ajusta perfectamente al desarrollo de nuestro problema.

La implementación que resuelve el problema planteado en este trabajo se ha dividido en 3 algoritmos que plantean desde una primera aproximación que nos ofrece relacionar la valoración de un atributo con los valores de una característica, hasta un algoritmo que permite relacionar los valores de un atributo con 'n' características diferentes.

Los algoritmos implementan funciones MapReduce cuyas entradas son el texto de cada fichero XML que representan la información de los estudios radiológicos.

Figura 13: Ejemplo de fichero de entrada XML

Lo primero que debemos hacer es descomponer el problema en trabajos tipo MapReduce. De manera general lo que deseamos es buscar todos los valores de un atributo contrastado con las características que lo definen. Establecemos los siguientes pasos para resolver el problema.

1. Leer el XML y buscar tanto el valor del atributo y el/las características que originan el valor. En este caso el programa MapReduce se encargará de crear la relación (atributo o valor, característica).
2. La parte de reducción toma los valores de salida de los mappers y define la salida final con el total de coincidencias encontradas.

La salida de la función de mapeo crea la relación (atributo, característica) y presenta de la siguiente forma (ver figura 13).

$$map\ (\ CompositeKey(valor\ de\ columna, valor\ de\ fila), contador\ )$$

Figura 14: Ejemplo función map

Ahora debemos tomar en cuenta otro aspecto de MapReduce, es que este ordena los registros por la clave antes de que estos lleguen a la fase de reduce. En nuestro caso en particular hay que tener en cuenta que en nuestro problema la clave es el par (atributo, característica) como se indica en la Figura 14. Si dejamos que el propio algoritmo ordene los registros en base a la manera explicada, los registros llegarían mal ordenados ya que cada registro proviene de diferentes mappers y sería un problema ya que se tendría que realizar una ordenación manual en la fase de reduce. Para evitar esto, existe una manera de realizar una ordenación secundaria.

Para que MapReduce realice una ordenación secundaria primero es necesario que definamos nuestra clave simple a una que sea compuesta. En nuestro caso nuestra nueva clave será la composición del valor del atributo y su característica. Después hará falta un crear un Agrupador (*GroupingComparator*) el cual se encargará de agrupar mismos valores de las claves a la misma llamada en el reducer.

6.1. Algoritmos implementados

Los algoritmos implementados se han desarrollado usando Java como lenguaje de programación, dado que Apache Hadoop es un proyecto construido en este lenguaje por lo que lo soporta de manera natural. Es posible realizar implementaciones en otros lenguajes ya que Hadoop viene con una utilidad llamada Hadoop Streaming [41], que permite crear y ejecutar trabajos MapReduce especificando cualquier programa o script como mappers y reducers que lean y escriban de/en la salida estándar.

Los algoritmos presentados en esta sección están disponibles en la sección *9. Anexos.* A continuación daremos una visión general de cómo se ha realizado la implementación.

Nivel 1

Esta primera aproximación de la solución implementa un algoritmo MapReduce en el que se relaciona los valores de las categorías BI-RADS contra una característica que puede ser cualquiera que tengamos disponible en el estudio como tipo de lesión, morfología, densidad, etc. La característica que queremos evaluar se define en un fichero **.**properties (ver Figura 15) que debe estar subido en el sistema de ficheros que vayamos a usar, sea HDFS o S3.

Este fichero tiene claves, llamadas XPathColumna y XPathFila, en las que deberemos definir el valor en formato XPATH de las características que deseamos evaluar. Como se habló en secciones anteriores la información proviene de documentos XML estructurados y, por lo tanto, la mejor manera de acceder a la información contenida en ellos y que a su vez permita una mayor flexibilidad a la hora de configurar las propiedades a evaluar es a través de XPATH [42] ya que es un lenguaje diseñado para recorrer y procesar un documento XML.

Figura 15: Trozo de ejemplo de fichero configuración.properties

Una vez empieza la fase de map, se leen los informes XML y se escriben los pares (atributo, característica). Por ejemplo ("BI-RADS 2", "Calcificaciones")

Durante la fase reduce, se contabilizan los resultados que se han encontrado y se muestra la salida final (ver Figura 16).

```
/MAMO/BI-RADS 1,/MAMO/Nódulo / Masa,1
/MAMO/BI-RADS 1,/MAMO/Calcificaciones,38
/MAMO/BI-RADS 1,/MAMO/Distorsión Arquitectural o Alteración Estructural,36
/MAMO/BI-RADS 2,/MAMO/Asimetrias,47
/MAMO/BI-RADS 2,/MAMO/Ganglio Linfático Intramamario,46
/MAMO/BI-RADS 2,/MAMO/Lesión de la Piel,32
/MAMO/BI-RADS 3,/MAMO/Conducto Dilatado Aislado,38
/MAMO/BI-RADS 3,/MAMO/Calcificaciones,51
/MAMO/BI-RADS 3,/MAMO/Conducto Dilatado Aislado,39
/MAMO/BI-RADS 4B,/MAMO/Nódulo / Masa,43
/MAMO/BI-RADS 4B,/MAMO/Calcificaciones,43
/MAMO/BI-RADS 4B,/MAMO/Lesión de la Piel,43
/MAMO/BI-RADS 4C,/MAMO/Nódulo / Masa,36
/MAMO/BI-RADS 4C,/MAMO/Calcificaciones,48
/MAMO/BI-RADS 4C,/MAMO/Conducto Dilatado Aislado,44
/MAMO/BI-RADS 4D,/MAMO/Nódulo / Masa,43
/MAMO/BI-RADS 4D,/MAMO/Calcificaciones,32
/MAMO/BI-RADS 4D,/MAMO/Conducto Dilatado Aislado,54
/MAMO/BI-RADS 5,/MAMO/Asimetrias,24
/MAMO/BI-RADS 5,/MAMO/Calcificaciones,4
/MAMO/BI-RADS 5,/MAMO/Conducto Dilatado Aislado,43
/MAMO/BI-RADS 6,/MAMO/Nódulo / Masa,159
/MAMO/BI-RADS 6,/MAMO/Ganglio Linfático Intramamario,38
/MAMO/BI-RADS 6,/MAMO/Conducto Dilatado Aislado,38
```

Figura 16: Ejemplo de salida de la fase reduce para algoritmo nivel 1

NIVEL 2

Esta aproximación ahora pretende aumentar la granularidad de los resultados ya que así podemos tener una visión más detallada. Ahora el algoritmo va a tomar el valor de un atributo BI-RADS y lo va a asociar a dos características "tipo lesión" y "morfología" por ejemplo. La clave compuesta tendrá una triada de atributos, uno para los valores y dos para las características que queremos comparar. Cabe recalcar que la misma manera que en los algoritmos anteriormente presentados, los atributos y características se leen de un fichero de configuración, con la diferencia que las 2 características se colocan separadas por coma en el valor de la variable XPathFila.

El funcionamiento de nuestro algoritmo sigue siendo el mismo, solo que ahora la salida en este nivel tiene mucho más información (ver Figura 17). Claramente se puede evidenciar que ahora tenemos mucho más detalle de información. Además introducimos

la posibilidad de contrastar información transversal entre los diferentes estudios en los informes. Es decir que ahora vamos poder contrastar la información de mamografías con la de resonancias magnéticas por ejemplo. Para lograr esto tendremos que colocar al inicio de cada variable XPATH el estudio del que obtendremos la información /MAMO /RESO /ECO para mamografías, resonancias y ecografías respectivamente. Con esto podremos ahora formar claves de la forma ("/MAMO/BI-RADS 2", "/MAMO/Calcificaciones", "/ECO/Oval"). Esto nos abre la posibilidad de cruzar la información que aparece en uno u otro estudio y que no aparece en otros pero que de igual forma aportan datos para el diagnóstico del cáncer de mama.

```
/MAMO/BI-RADS 1,/MAMO/Nodulo / Masa,/ECO/Oval,  1
/MAMO/BI-RADS 1,/MAMO/Nodulo / Masa,/ECO/Estrellado,    1
/MAMO/BI-RADS 1,/MAMO/Nodulo / Masa,/ECO/Redondo,   4828
/MAMO/BI-RADS 1,/MAMO/Asimetrias,/ECO/Redondo,  4772
/MAMO/BI-RADS 1,/MAMO/Calcificaciones,/ECO/Redondo, 4763
/MAMO/BI-RADS 1,/MAMO/Conducto Dilatado Aislado,/ECO/Redondo,   4843
/MAMO/BI-RADS 1,/MAMO/Distorsión Arquitectural o Alteración Estructural,/ECO/Redondo,   4707
/MAMO/BI-RADS 1,/MAMO/Ganglio Linfatico Intramamario,/ECO/Redondo,  4811
/MAMO/BI-RADS 1,/MAMO/Lesion de la Piel,/ECO/Redondo,   4732
/MAMO/BI-RADS 1,/MAMO/Protuberancias,/ECO/Irregular,    3
/MAMO/BI-RADS 1,/MAMO/Protuberancias,/ECO/Oval, 2
/MAMO/BI-RADS 2,/MAMO/ Masa,/ECO/Oval,  1
/MAMO/BI-RADS 2,/MAMO/ Masa,/ECO/Estrellado,    1
/MAMO/BI-RADS 2,/MAMO/ Masa,/ECO/Redondo,   4666
/MAMO/BI-RADS 2,/MAMO/Asimetrias,/ECO/Redondo,  4746
/MAMO/BI-RADS 2,/MAMO/Calcificaciones,  300088
/MAMO/BI-RADS 2,/MAMO/Calcificaciones,/ECO/Irregular,   1
/MAMO/BI-RADS 2,/MAMO/Calcificaciones,/ECO/Oval,    2
/MAMO/BI-RADS 2,/MAMO/Calcificaciones,/ECO/Estrellado,  1
/MAMO/BI-RADS 2,/MAMO/Calcificaciones,/ECO/Redondo, 4737
/MAMO/BI-RADS 2,/MAMO/Conducto Dilatado Aislado,/ECO/Redondo,   4631
/MAMO/BI-RADS 2,/MAMO/Distorsión Arquitectural o Alteración Estructural,/ECO/Redondo,   4583
/MAMO/BI-RADS 2,/MAMO/Ganglio Linfatico Intramamario,/ECO/Redondo,  4750
/MAMO/BI-RADS 2,/MAMO/Lesion de la Piel,/ECO/Redondo,   4790
/MAMO/BI-RADS 2,/MAMO/Protuberancias,/ECO/Oval, 2
/MAMO/BI-RADS 2,/MAMO/Protuberancias,/ECO/Estrellado,   2
/MAMO/BI-RADS 2,/MAMO/Protuberancias,/ECO/Redondo,  1
/MAMO/BI-RADS 3,/MAMO/ Masa,/ECO/Irregular, 1
/MAMO/BI-RADS 3,/MAMO/ Masa,/ECO/Oval,  2
/MAMO/BI-RADS 3,/MAMO/ Masa,/ECO/Estrellado,    1
/MAMO/BI-RADS 3,/MAMO/ Masa,/ECO/Redondo,   4774
```

Figura 17: Ejemplo de salida de la fase reduce para algoritmo nivel 2

NIVEL N

En este algoritmo se ofrece una flexibilidad total a la hora de escoger las características que vamos a extraer. En este nivel la clave compuesta va a cambiar de manera sustancial ya que se debe permitir almacenar N características. Por lo tanto ahora la clave compuesta estará definida por un listado de características que permitirá guardar el número de características que deseemos. De igual manera que en los anteriores algoritmos, las características se configuran en el fichero **.**properties y separados por coma. También se permite el análisis de información transversal entre informes y con esto podremos lograr un nivel de detalle en que se puedan contrastar todas las características de las lesiones de todos informes.

6.2. Visualización de Resultados

Para extraer el conocimiento que deseamos vamos a usar R en conjunto con el paquete *ggplot2* [40] que es una librería que permite crear gráficos complejos para análisis de datos.

Una vez que se ha terminado de procesar el/los SequenceFiles que contienen los XML con los informes médicos, tenemos como resultado un fichero de texto (ver figura 17) que en sí mismo no es muy útil. Este fichero lo vamos a procesar con R y la librería ggplot2 para generar un gráfico que permita identificar los patrones de forma visual.

Para el algoritmo de nivel 1, sólo tenemos la comparación de un valor, que son los BI-RADS contra una propiedad o característica. El resultado es bastante simple pero ya deja ver qué clase información podremos recolectar (ver Figura 18).

Ahora todos los miles de informes que estaban dispersos, los tenemos condensados en un gráfico que resume la información que en ellos están, permitiendo a una persona interpretarlo de forma sencilla.

Sin ser muy entendidos en el ámbito de la medicina, observando el gráfico (ver Figura 18) podríamos decir que la lesión Nódulo / Masa en la categoría BI-RADS 4B. Incluso se puede especular que la lesión Nódulo/Masa tiene tendencia a ser de carácter maligno debido a que aparece en otras categorías de BI-RADS como la 5 y la 6 donde las posibilidades de que la lesión sea considerada como un cáncer son más altas.

Finalmente tenemos que en la categoría BI-RADS 6 que supone confirmaciones mediante biopsias de que se trata de cáncer. Y nuevamente encontramos la lesión Nódulo/Masa la cual se confirma que son lesiones cancerígenas. Entonces la pregunta que puede surgir es ¿por qué tenemos lesiones Nódulo/masa en varias categorías BI-RADS diferentes? Si ya se ha confirmado en la categoría BI-RADS 6 que las lesiones Nódulo / Masa son lesiones cancerígenas. Necesitamos de algo más de detalle que nos clarifique el panorama de los resultados del estudio.

Hasta aquí, la interpretación que hemos dado a los resultados mostrados en el nivel 1 de nuestro algoritmo, parecen un poco simple y carente de mayor detalle como lo hemos podido notar a la hora de querer dar una interpretación a los mismos.

Ahora podemos usar un poco más de granularidad usando el algoritmo de nivel 2 para ver si es posible obtener mayor detalle.

En la Figura 19, podemos ver los resultados de nuestro caso de estudio de los BI-RADS aplicando el algoritmo de nivel 2. Para la ejecución del algoritmo se han tomado como características el tipo de lesión y la forma (tomado de las ecografías). Ahora con un mejor nivel de detalle podemos responder la pregunta que hicimos al mostrar los resultados con un algoritmo de nivel 1. Ahora se puede observar que las lesiones Nódulo/Masa que se han confirmado como un cáncer en la categoría BI-RADS 6 son de forma redonda y los que están categorizados como BI-RADS 4 son Nódulos/Masa de forma estrellada.

A medida que apliquemos al caso de estudio de los BI-RADS un mayor nivel de detalle podremos tener un panorama mucho más claro de porqué una determinada lesión es encasillada dentro de una categoría BI-RADS.

Figura 18: Gráfico de estadísticas para el nivel 1

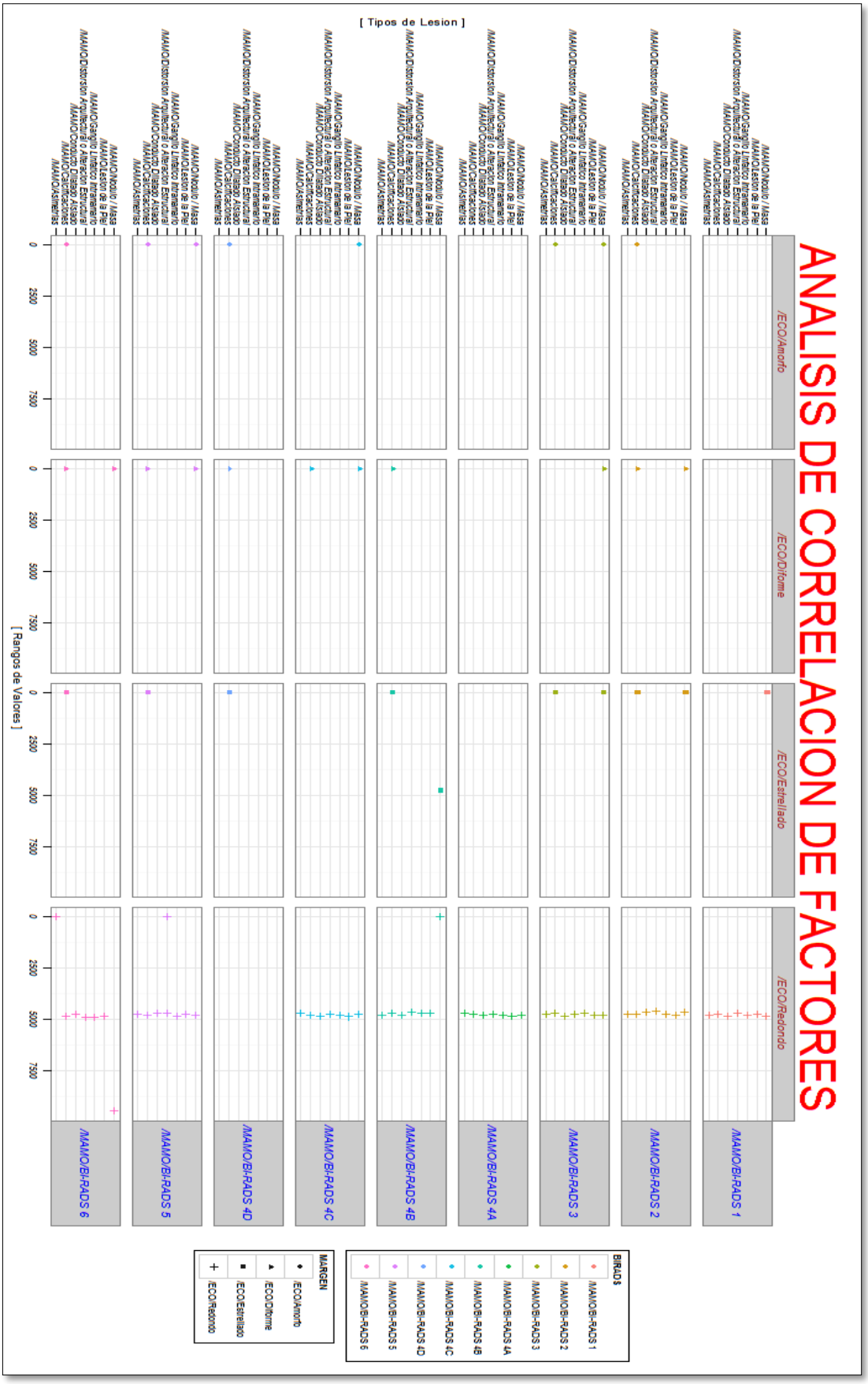

Figura 19: Gráfico de estadísticas para el nivel 2

6.3. Análisis de recursos

Una vez que hemos realizado la implementación que resuelve el problema planteado y además de verificar su correcto funcionamiento, podemos realizar un pequeño estudio de las características de rendimiento que supone utilizar nuestra solución al estudio de los BI-RADS.

Las mediciones a realizar implican realizar predicción de nodos en el cluster y como se podría determinar los requerimientos de memoria y CPU de nuevos nodos para añadir en el cluster, en el caso de necesitarlo, lo cual también dependerá del tamaño de los ficheros que usaremos para la muestra.

En nuestro caso usaremos un fichero SequenceFile de tamaño de alrededor de 800MB que contiene empaquetados 300.000 informes de cáncer de mama, cada uno con tres estudios de mamografías, resonancias y ecografías. La creación de este fichero no forma parte del presente análisis ya que estos ficheros se crearían una sola vez y se los utilizaría en múltiples estudios "Write once read many".

Debemos conocer como Hadoop tratará el fichero para generar los mappers y así podremos tener una visión de número de procesos paralelos que Hadoop podrá lanzar a la hora de ejecutar nuestro trabajo. MapReduce tomará el fichero de 800MB que hemos generado y lo dividirá en N tareas repartidas dentro de los nodos del cluster y que tendrán que procesar ficheros de menor tamaño. De hecho, el tamaño a procesar en cada subproceso es de 64MB. Además es fácil deducir que a mayor número de procesos, el tiempo lineal de procesamiento total también será menor.

De una manera sencilla podemos saber cuál es el número de mappers que se van a lanzar para procesar los 800MB de nuestro fichero. Si sabemos que el tamaño de bloque está configurado en 64MB (que es el valor por defecto), solo tendremos que dividir 800MB entre 64MB, lo cual nos dará un valor de 12.5. En otras palabras tendremos 13 mappers, 12 que procesarán bloques completos de 64MB y uno que tendrá un bloque con menos información.

Al momento de enviar nuestro trabajo a nuestro cluster hadoop y este lo empiece a procesar, veremos en los logs (ver Figura 20) que se indica en cuantos mappers se ha dividido el trabajo.

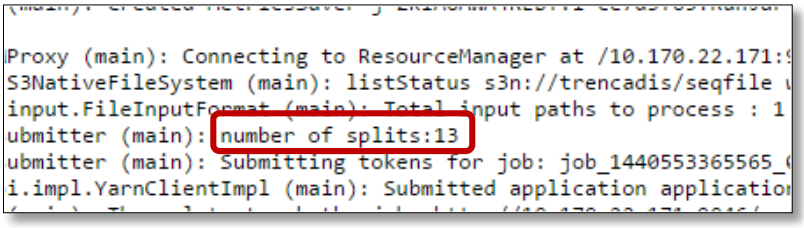

Figura 20: Salida de logs de hadoop con el número partes en las que se dividido un fichero

También es posible saber esta información, al final de la ejecución donde se ofrecen un resumen de diferentes contadores (ver Figura 21).

```
Job Counters
    Launched map tasks=13
    Launched reduce tasks=1
    Data-local map tasks=13
    Total time spent by all maps in occupied slots (ms)=4691598
    Total time spent by all reduces in occupied slots (ms)=51644
    Total time spent by all map tasks (ms)=1563866
    Total time spent by all reduce tasks (ms)=12911
    Total vcore-seconds taken by all map tasks=1563866
    Total vcore-seconds taken by all reduce tasks=12911
    Total megabyte-seconds taken by all map tasks=1201049088
    Total megabyte-seconds taken by all reduce tasks=13220864
```

Figura 21: Salida de contadores con el número de mappers generados

La información que hemos sacado es bastante útil dado el ambiente en el que estamos trabajando, el cloud, donde usamos infraestructuras de pago por uso. El conocer con antelación cuantos procesos paralelos se pueden llegar a necesitar nos permite parametrizar nuestro cluster de manera que podamos lanzar el número preciso de nodos y así no incurrir en costos innecesarios. Para nuestro caso, y después del análisis hecho, hemos lanzado un cluster con 7 DataNodes dado que cada instancia que hace de DataNode puede ejecutar dos mappers de forma paralela [30]. Si configuramos el número correcto de datanodes tendremos también el máximo nivel de paralelismo y por lo tanto el menor tiempo lineal.

Hemos concluido que para nuestro ejercicio, el número óptimo de DataNodes en donde se alcanza el máximo paralelismo y se optimiza la ejecución es 8 (7 DataNodes + 1 MasterNode), es decir, que a partir de este número de nodos no vamos a obtener ninguna mejora con la inclusión de más DataNodes y es más, estaremos incurriendo en costos innecesarios. La Figura 22 muestra el sitio de administración ResourceManager con el número de nodos que forman un cluster Hadoop.

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
1	0	0	1	0	0 B	26 GB	0 B	13	0	0	0	0

Show 20 entries Search:

Rack	Node State	Node Address	Node HTTP Address	Last health-update	Health-report	Containers	Mem Used	Mem Avail	Version
/default-rack	RUNNING	ip-10-182-164-48.ec2.internal:9103	ip-10-182-164-48.ec2.internal:9035	26-Aug-2015 02:31:11		0	0 B	2 GB	2.4.0-amzn-1
/default-rack	RUNNING	ip-10-182-128-198.ec2.internal:9103	ip-10-182-128-198.ec2.internal:9035	26-Aug-2015 02:30:58		0	0 B	2 GB	2.4.0-amzn-1
/default-rack	RUNNING	ip-10-232-74-84.ec2.internal:9103	ip-10-232-74-84.ec2.internal:9035	26-Aug-2015 02:31:05		0	0 B	2 GB	2.4.0-amzn-1
/default-rack	RUNNING	ip-10-182-142-36.ec2.internal:9103	ip-10-182-142-36.ec2.internal:9035	26-Aug-2015 02:30:57		0	0 B	2 GB	2.4.0-amzn-1
/default-rack	RUNNING	ip-10-235-25-88.ec2.internal:9103	ip-10-235-25-88.ec2.internal:9035	26-Aug-2015 02:31:04		0	0 B	2 GB	2.4.0-amzn-1
/default-rack	RUNNING	ip-10-159-67-55.ec2.internal:9103	ip-10-159-67-55.ec2.internal:9035	26-Aug-2015 02:30:56		0	0 B	2 GB	2.4.0-amzn-1
/default-rack	RUNNING	ip-10-232-67-23.ec2.internal:9103	ip-10-232-67-23.ec2.internal:9035	26-Aug-2015 02:30:52		0	0 B	2 GB	2.4.0-amzn-1
/default-rack	RUNNING	ip-10-235-0-211.ec2.internal:9103	ip-10-235-0-211.ec2.internal:9035	26-Aug-2015 02:31:06		0	0 B	2 GB	2.4.0-amzn-1
/default-rack	RUNNING	ip-10-235-7-113.ec2.internal:9103	ip-10-235-7-113.ec2.internal:9035	26-Aug-2015 02:30:58		0	0 B	2 GB	2.4.0-amzn-1
/default-rack	RUNNING	ip-10-171-20-104.ec2.internal:9103	ip-10-171-20-104.ec2.internal:9035	26-Aug-2015 02:29:15		0	0 B	2 GB	2.4.0-amzn-1
/default-	RUNNING	ip-10-235-31-	ip-10-235-31-	26-Aug-2015		0	0 B	2 GB	2.4.0-

Figura 22: Resumen del Resource Manager con el número de nodos que forman el cluster de EMR

De hecho es posible entrar en la información del Cluster y ver cuánto tiempo se ha demorado nuestro trabajo en procesar toda la información. Para procesar los 300.000 informes se han necesitado sólo 4 minutos (ver Figura 23).

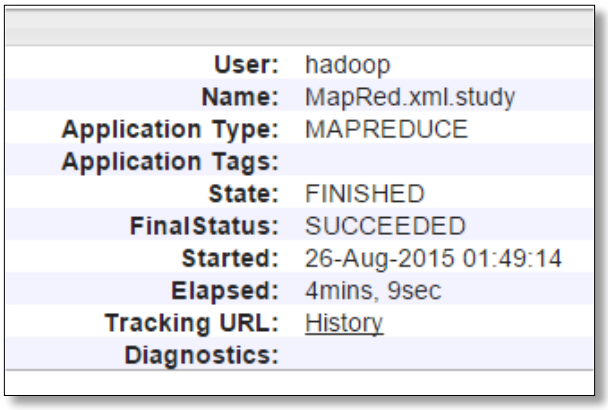

Figura 23: Resumen con el estado final del trabajo

Hasta ahora hemos medido el problema desde el punto de vista de la eficiencia en tiempos y procesamiento. Pero debido a que estamos utilizando infraestructuras de pago por uso, es importante tener en cuenta el factor económico no sólo por el número de instancias a usar en el cluster sino también por el tipo de características que tengan dichas instancias, ya que mientras más recursos tengan mayor será su coste por unidad de tiempo.

Si necesitamos lanzar un número N determinado de instancias en el cluster hadoop, estás nos generarán un coste que va a depender de la capacidad de cálculo que tenga cada máquina. Por lo que es interesante calcular las necesidades de recursos que necesitarían nuestras instancias de EC2.

El procedimiento que ahora se va a aplicar no es sencillo, y tampoco menester de este trabajo enseñar o detallar el mismo de una manera extensa y profunda. Una vez que tenemos desplegado nuestro cluster Hadoop, desplegado mediante Amazon EMR, vamos conectar por medio de SSH al nodo máster. Al momento de desplegar el cluster, nuestro par de claves serán inyectados en los nodos del cluster de manera que nos podremos conectar sin contraseña. Una vez dentro del nodo máster obtendremos la lista de los DataNodes. Esto lo podemos conocer de varias maneras, podemos ver en la Figura 22 que dentro de la etiqueta "Node Address" se encuentra la IP de cada DataNode, podríamos simplemente tomar una de ellas y desde el máster conectarnos a esa IP. Otra opción sería usar el comando $hadoop\ dfsadmin - report$ en el cual obtendremos un listado de los nodos que conforman el cluster.

Vamos a utilizar una herramienta para realizar profiling en Java que se llama VisualVM [43]. Esta herramienta nos ofrece una interfaz para obtener información detallada tanto de manera visual como a través de línea de comando para la monitorización y rendimiento de aplicaciones Java. Cada Mapper que se lanza en Hadoop se ejecuta sobre su propia máquina virtual. Además VisualVM permite monitoizar procesos remotos por lo que este procedimiento lo vamos a realizar desde el nodo master.

Para poder visualizar tanto las interfaces web de nuestro cluster hadoop como las herramientas que vamos a utilizar, es necesario habilitar SSH X11 forwarding y crear un túnel SSH entre la máquina de trabajo local y el nodo máster [44].

Ya en el nodo máster, mientras se ejecuta nuestro trabajo MapReduce, lanzaremos VisualVM y lo enlazaremos al proceso Java que dicho DataNode, que al final es un mapper que se ha lanzado en su propia máquina virtual. Una vez enlazado el proceso, VisualVM empezará a recolectar varias estadísticas de rendimiento, consumo de memoria, número de threads, etc.

En este caso, para determinar las características de recursos que requieren las instancias EC2 que se deben lanzar en un cluster hadoop de EMR, nos interesa es el de rendimiento y consumo de memoria.

Este estudio se basa en un cluster generado con máquinas m1.medium, que es la instancia recomendada que indica Amazon EMR.

Las características en cuanto a recursos para el tipo de instancia utilizada, se encuentran en el siguiente listado.

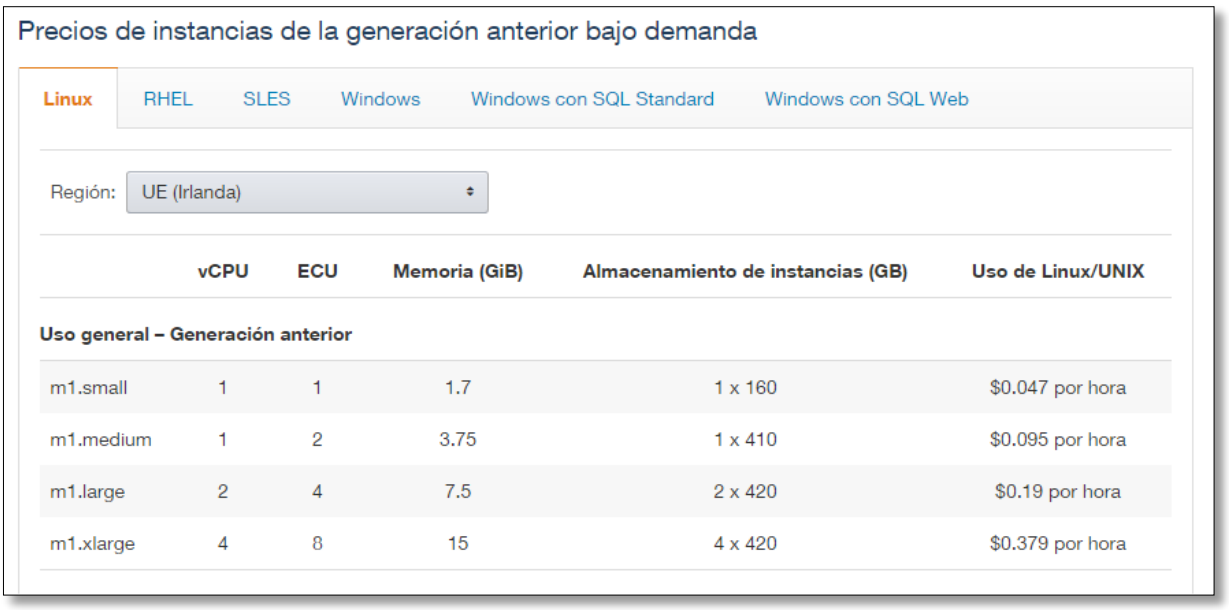

Figura 24: Listado de recursos de instancias m1 de AWS [45]

Como se puede observar en la Figura 24, hemos realizado el estudio con máquinas con 1 core y 3.75GB de RAM cuyo costo es 0.095 USD por hora.

En cuanto a los resultados, tenemos que en cuanto a rendimiento de CPU, el estudio nos indica que para nuestro ejercicio, con máquinas de 1 core, el uso de CPU durante el intervalo de ejecución del trabajo estuvo alrededor del 50% (ver Figura 25).

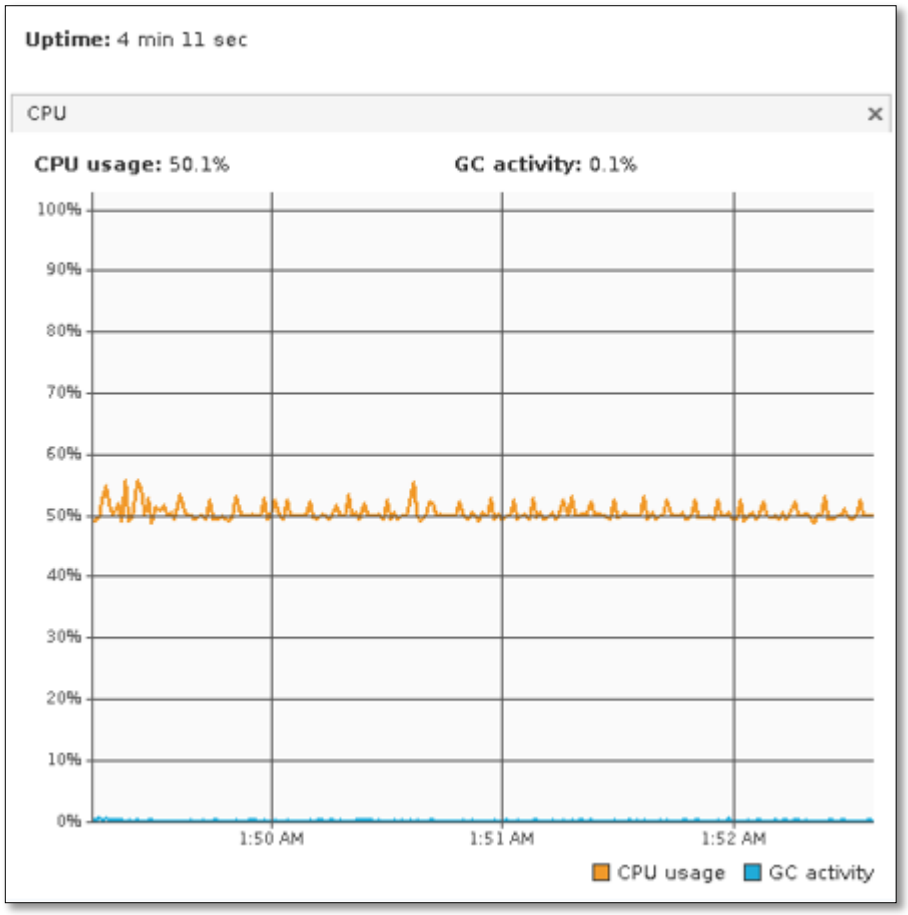

Figura 25: Uso de CPU

En cuanto a la memoria, tenemos que indicar que se está midiendo el *Heap*, que es el espacio de memoria que se utiliza para almacenar clases, objetos y arreglos. Hemos utilizado instancias EC2 que usan 3.75GB de memoria RAM. El resultado del análisis del consumo de memoria nos indica que durante la ejecución del trabajo MapReduce, el heap alcanzó un máximo de uso de 350MB y su tamaño máximo era de alrededor de 536MB. Este resultado nos muestra que los procesos que se envían a los Datanodes no requieren de mayor memoria RAM para el heap de Java. Entonces si dispusiéramos de instancias con menos memoria pero con la misma cantidad de cores podríamos hacer uso de ellas y reduciríamos costos porque costarán menos (ver Figura 26).

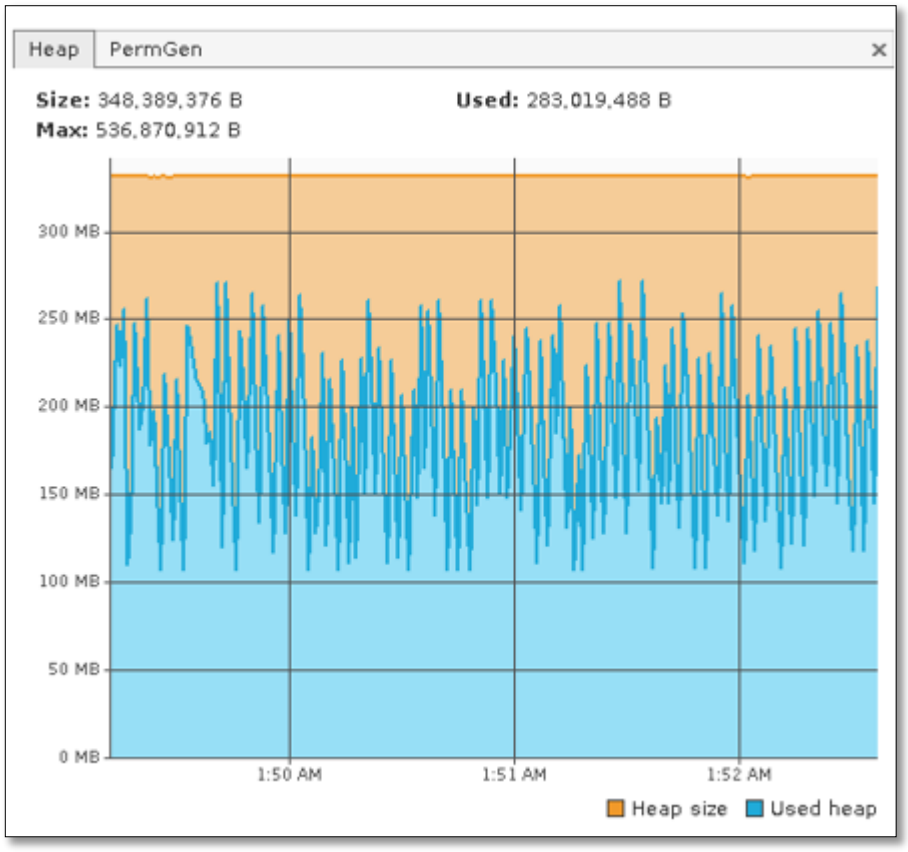

Figura 26: Uso de memoria

Las conclusiones que podemos sacar de este pequeño estudio, son orientativos a tratar de sacar el máximo provecho tanto en la potencia de cálculo que podemos sacar como a optimizar los costos que podemos tener dado que usamos infraestructuras de pago por uso. Es una guía que ha partido de un conjunto de datos específicos y que se adaptan al ejemplo planteado, otros conjuntos de datos u otro tipo de ejecución puede diferir en la información resultante.

Sin embargo el estudio en sí, se puede aplicar para identificar que los pasos necesarios y herramientas a utilizar para identificar el tipo hardware y sus características para resolver el mismo problema pero a mayor escala o problemas con características similares utilizando el menor tiempo posible y a su vez optimizando los costos.

7. Conclusiones y trabajos futuros

A través de este trabajo hemos podido analizar la factibilidad de usar algoritmos MapReduce en el tratamiento y análisis de factores de relación en informes médicos relacionados al cáncer de mama.

Se ha mostrado la viabilidad de usar AWS un proveedor de Cloud público por medio del servicio Amazon EMR, sin perjuicio de uso de otras infraestructuras on-premises. Se han analizado los diferentes métodos que ofrece HDFS para el almacenamiento de ficheros. De manera particular se ha analizado el problema de los ficheros pequeños dado los requerimientos del problema y así el sistema de almacenamiento que mejor se ajuste.

Se han estudiado los diferentes tipos de almacenamiento que ofrece HDFS como los HAR files, los MapFiles y los SequenceFiles, siendo el último, el método de almacenamiento elegido como contenedor de la información.Se ha desarrollado algoritmos MapReduce que permiten la resolución del caso de estudio de los BI-RADS desde la computación paralela. Los resultados obtenidos con la aplicación de estos algoritmos nos reafirman que son válidos para estudios en el área de la informática médica.

Se ha realizado un estudio de la ejecución en un cluster Hadoop y se ha analizado el consumo de memoria Java. Esto es especialmente útil cuando se utilice una plataforma elástica como Amazon EMR que nos permite añadir nodos bajo demanda. Con este estudio podremos conocer de antemano cuales son los requerimientos de memoria y CPU que requiere un DataNode y saber qué tipo de instancias se deben lanzar y así no incurrir en costos innecesarios.

Se ha utilizado una herramienta para el análisis estadístico muy usada en el ámbito científico como lo es R. A través de este lenguaje hemos logrado obtener gráficas de gran calidad que permitirán analizar de manera correcta los resultados obtenidos.

Una posible ampliación a este trabajo sería trabajar con otras fuentes de información, no sólo el estudio en informes de cáncer de mama, sino cualquier tipo de estudio de la cual se quiera extraer algún conocimiento. Además de considerar otro tipo de formas de estructurar la información, no solo XML.

8. Bibliografía

1. Channin DS, Mongkolwat P, Kleper V, Rubin DL. Computing human image annotation. Conf Proc IEEE Eng Med Biol Soc 2009; 1:7065–8.
2. Chhatwal J, Alagoz O, Lindstrom MJ, Kahn Jr CE, Shaffer KA, Burnside ES. A logistic regression model based on the national mammography database format to aid breast cancer diagnosis. AJR Am J Roentgenol. 2009;192: 1117---27.
3. Juan Carlos Andrade Castañeda. Estudio de los beneficios de la implantación del modelo de gestión cloud computing en comparación al modelo de gestión tradicional. Pp 78 – 81.
4. Amazon Web Services. What is Cloud Computing? Disponible en http://aws.amazon.com/es/what-is-cloud-computing/
5. IBM Web Site. Hadoop. What is Mapreduce? Disponible en http://www-01.ibm.com/software/data/infosphere/hadoop/mapreduce/
6. Innovación, Big Data y Epidemiología. Revista Iberoamericana de argumentación. RIA 7 (2013): 1-12
7. Dromain C, Boyer B, Ferré R, Canale S, Delaloge S, Balleyguier C. Computed-aided diagnosis (CAD) in the detection of breast cancer. Eur J Radiol. 2012 Aug 29
8. American Cancer Society. Disponible en: http://www.cancer.org/espanol/servicios/comocomprendersudiagnostico/fragmentado/mamogramas-y-otros-procedimientos-de-los-senos-con-imagenes-mammo-report (Consultado el 5 de agosto de 2015)
9. D. A. Clunie - DICOM Structured Reporting. PixelMed Publishing. Bangor, Pennsylvania. 2000.
10. R. Medina, E. Torres, D. Segrelles, I. Blanquer, L.Martí, D. Almenar. A Systematic Approach for Using DICOM Structured Reports in Clinical Processes: Focus on Breast Cancer.
11. Shin JH, Han BK, Ko EY, Choe YH, Nam SJ. Probably benign breast masses diagnosed by sonography: is there a difference in the cancer rate according to palpability? AJR Am J Roentgenol. 2009;192:W187---91.
12. Park YM, Kim EK, Lee JH, Ryu JH, Han SS, Choi SJ, et al. Palpable breast masses with probably benign morphology at sonography: can biopsy be deferred? Acta Radiol. 2008;49:1104---11. 22
13. Graf C, et al. Follow-up of palpable circumscribed noncalcified solid breast masses at mammography and US: can biopsy be averted? Radiology. 2004;233:850---6.
14. Kim EK, Ko KH, Oh KK, Kwak JY, You JK, Kim MJ, et al. Clinical application of the BI-RADS final assessment to breast sonography in conjunction with mammography. AJR Am J Roentgenol. 2008;190:1209---15.
15. Chhatwal J, Alagoz O, Lindstrom MJ, Kahn Jr CE, Shaffer KA, Burnside ES. A logistic regression model based on the national mammography database format to aid breast cancer diagnosis. AJR Am J Roentgenol. 2009;192: 1117---27.
16. Hadoop Web Page, The Apache Software Foundation. 07/07/2015. Disponible en: http://hadoop.apache.org. Consultado el 5 de Agosto de 2015)

17. IBM Web Site. Hadoop. What is Mapreduce? Disponible en http://www-01.ibm.com/software/data/infosphere/hadoop/mapreduce/
18. B. Revert. DICOM Cookbook. Philips Medical System Nederland, 1997
19. H. K. Huang. "PACS and Imaging Informatics: Basic Principles and Applications". ISBN: 0471251232. Editorial Wiley-Liss.
20. R. Noumeir - Benefits of the DICOM Structured Report. Journal of Digital Imaging. December 2006, Volume 19, Issue 4, pp 295-306.
21. A. Lith, J. Mattsson - Investigating storage solutions for large data. Department of Computer Science and Engineering. Chalmers University Of Technology. Göteborg, Sweden, 2010
22. Hortonworks Web Page. 2015. Disponible en: http://hortonworks.com/ (Consultado el 01 de agosto de 2015)
23. Cloudera Web Page. 2015. Disponible en: www.cloudera.com (Consultado el 01 de agosto de 2015)
24. Hadoop Web Page. Disponible en http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html
25. HortonWorks Web Page. What HDFS does. Disponible en http://hortonworks.com/hadoop/hdfs/
26. IBM BigInsights Web Page. 2015. Disponible en: http://www-01.ibm.com/software/data/infosphere/hadoop/enterprise.html (Consultado el 05 de agosto de 2015)
27. HDInsights Web Page. 2015. Disponible en: http://azure.microsoft.com/es-es/services/hdinsight/ (Consultado el 05 de agosto del 2015)
28. Amazon EMR. 2015. Disponible en: https://aws.amazon.com/es/elasticmapreduce/ (Consultado el 05 de agosto del 2015).
29. Google Compute Engine. Disponible en https://cloud.google.com/compute/
30. Amazon EMR Best Practices. Disponible en https://media.amazonwebservices.com/AWS_Amazon_EMR_Best_Practices.pdf. (Consultado el 31 de agosto del 2015)
31. Cloudera Web Page. Hadoop I/O: Sequence, Map, Set, Array, BloomMap Files. Disponible en http://blog.cloudera.com/blog/2011/01/hadoop-io-sequence-map-set-array-bloommap-files/
32. Hadoop Wiki: SequenceFile. 2015. Disponible en: http://wiki.apache.org/hadoop/SequenceFile (Consultado el 10 de agosto de 2015)
33. Tom White. 2012. Hadoop - The Definitive Guide; Third Edition. Pp 132 – 140
34. The Small Files Problem. Disponible en: http://blog.cloudera.com/blog/2009/02/the-small-files-problem/ (Consultado el 10 de agosto de 2015)
35. The Small Files Problem. Disponible en: http://inquidia.com/news-and-info/working-small-files-hadoop-part-1 (Consultado el 29 de Julio de 2015)
36. Yahoo Developer Network. Hadoop Archive: File Compaction for HDFS. 2015. Disponible en https://developer.yahoo.com/blogs/hadoop/hadoop-archive-file-compaction-hdfs-461.html (Consultado el 18 de agosto de 2015)

37. R Web Page. 2015. Disponible en: https://www.r-project.org/ (Consultado el 15 de agosto del 2015).
38. Tableau Web Page. Disponible en http://www.tableau.com/es-es (Consultado el 1 de septiembre de 2015)
39. GnuPlot Web Page. Disponible en http://www.gnuplot.info/ (Consultado el 1 de septiembre de 2015)
40. Ggplot2 Web Page. 2015. Disponible en: http://ggplot2.org/ (Consultado el 18 de agosto de 2015)
41. Hadoop Streaming, Hadoop Web Page. Disponible en http://hadoop.apache.org/docs/r1.2.1/streaming.html#Hadoop+Streaming (Consultado el 7 de septiembre de 2015)
42. W3C XML Path Languaje (XPath). Revisado 7 septiembre 2015. Disponible en http://www.w3.org/TR/xpath/ (Consultado el 7 de septiembre 2015)
43. VisualVM Web Page. 2015. Disponible en: https://visualvm.java.net/ (Consultado el 03 de agosto de 2015)
44. ElasticMapReduce Developer Guide. Disponible en http://docs.aws.amazon.com/ElasticMapReduce/latest/DeveloperGuide/emr-ssh/tunnel-local.html.
45. Amazon Web Services. 2015. Disponible en http://aws.amazon.com/es/ec2/pricing/ (Consultado el 30 de agosto de 2015).

9. Anexos

9.1. Creación de los sequence files

```
package test10.nivel2;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.BytesWritable;
import org.apache.hadoop.io.SequenceFile;
import org.apache.hadoop.io.SequenceFile.Metadata;
import org.apache.hadoop.io.SequenceFile.Writer;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.compress.DefaultCodec;

public class SequenceFileWriter {

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

		// hadoop jar TestingProject.jar test10.nivel2.SequenceFileWriter /home/biadmin/TFM/redhat
/TFM/input/sequence_file.sf
		// hadoop jar TestingProject.jar test10.nivel2.SequenceFileWriter /home/hduser/xml
/home/hduser/sequence_file.sf

		// output
```

```
                String uri = args[1];
                Path path = new Path(uri);

                // input local file
                File infile = new File(args[0]);
                File fileList[] = infile.listFiles();

                Configuration conf = new Configuration();
                FileSystem fs = FileSystem.get(conf);
                fs.delete(path, true);

                Text key = new Text();
                BytesWritable value = new BytesWritable();

                SequenceFile.Writer writer = null;
                StringBuilder builder = new StringBuilder();

                System.out.println("Cuantos archivos: " + fileList.length);
                long start = System.currentTimeMillis();
                long stop;

                try {

                        writer = SequenceFile.createWriter(conf, Writer.file(path),
                                        Writer.keyClass(key.getClass()),
                                        Writer.valueClass(value.getClass()),
                                        Writer.bufferSize(fs.getConf().getInt("io.file.buffer.size",4096)),
                                        Writer.replication(fs.getDefaultReplication()),
                                        Writer.blockSize(1073741824),
                                        Writer.compression(SequenceFile.CompressionType.BLOCK, new
DefaultCodec()),
                                        Writer.progressable(null),
                                        Writer.metadata(new Metadata()));

                        for (File arch : fileList) {

                                if (arch.isFile() && arch.getName().endsWith(".xml")) {

                                        BufferedReader reader = new BufferedReader(new FileReader(arch));

                                        for(String line=reader.readLine(); line!=null; line=reader.readLine()) {
                                                builder.append(line.trim());
                                        }

                                        key = new Text();
                                        value = new BytesWritable();

                                        key.set(arch.getName());

        value.set(builder.toString().trim().getBytes(),0,builder.toString().trim().getBytes().length);
                                        writer.append(key, value);

                                        reader.close();
                                        reader = null;
                                        key = null;
                                        value = null;
                                        builder.delete(0, builder.length());
                                }
                        }

                        stop = System.currentTimeMillis();
                        System.out.println("[INFO] - " + (stop - start) + " milisegundos");

                } finally {
                        if (writer!=null) writer.close();
          }

        }
}
```

9.2. Extracción de información de los sequence files – Mapper

```
package test10.nivel2;

import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.nio.charset.Charset;
import java.util.ArrayList;
import java.util.List;
import java.util.Properties;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.BytesWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import org.dom4j.Document;
import org.dom4j.Element;
import org.dom4j.Node;


public class TheMapper extends Mapper<Object, BytesWritable, CompositeKey, LongWritable/*Text*/> {

	private String xpathColumna;
	private String xpathFila;
	private String[] pathsFila;
	private String[] pathEnding;
	private String[] pathInit;

	private static LongWritable one = new LongWritable(1);
	private Heartbeat heartBeat;

	/**
	 * SETUP
	 */
	@Override
	protected void setup(Mapper<Object, BytesWritable, CompositeKey, LongWritable/*Text*/>.Context context)
throws IOException, InterruptedException {

		heartBeat = Heartbeat.createHeartbeat(context);

		Configuration configuration = context.getConfiguration();

		Path pt = new Path(configuration.get("map.configuration.file")); // Esto es en HDFS -->
/TFM/configuracion.properties

		System.out.println("ARCHIVO DE PROPIEDADES: " + pt.getName() + " - " + pt.toUri());
		System.out.println("DEFAULT CHARSET: " + Charset.defaultCharset() + " - " +
getDefaultCharSet());

		FileSystem fileSystem = FileSystem.get(configuration);

		FSDataInputStream dataInputStream = fileSystem.open(pt);

		Properties propiedades = new Properties();
		InputStreamReader entrada = new InputStreamReader(dataInputStream, "ISO8859_9");
		propiedades.load(entrada);

	xpathColumna = propiedades.getProperty("XPathColumna1");

	System.out.println("Caracteristicas a analizar: " + xpathColumna);

	xpathFila = propiedades.getProperty("XPathFila1");
	pathsFila = xpathFila.split(";");
```

```
                int index = 0;
                pathEnding = new String[pathsFila.length];
                pathInit = new String[pathsFila.length];

                for (String path : pathsFila) {
                                pathEnding[index] = path.split("\\.")[1];
                                pathInit[index] = path.split("\\.")[0];
                                System.out.println("Caracteristicas a analizar: " + pathEnding[index]);
                                index++;
                        }

                super.setup(context);
            }


            @SuppressWarnings("unchecked")
            public void map(Object key, BytesWritable value, Context context) throws IOException, InterruptedException
{

                        String xml = MAMOUtils.fromBytesWritableToString(value);

                        if (xml == null || xml.equals(""))
                                return;

                        Document document = MAMOUtils.getXmlFromText(xml.trim());

                        if(document != null)
                        {

                                Element rootElement = document.getRootElement();
                                List<Node> nodosColumna = rootElement.selectNodes( xpathColumna );

                                List<Text> valores = null;
                                for (int i = 0; i < nodosColumna.size(); i++)
                                {
                                        Node columna = nodosColumna.get(i).getParent();

                                        valores = new ArrayList<Text>();
                                        CompositeKey compositeKey = new CompositeKey();
                                        Text textColumna = new Text(xpathColumna + "/" +
columna.selectSingleNode("VALUE/CODE_MEANING").getText());
                                        compositeKey.setColumna( new
Text(createAliasForXPath(textColumna.toString())) );

                                        Node lesion = nodosColumna.get(i).getParent().getParent();
                                        document = MAMOUtils.getXmlFromText( lesion.asXML() );

                                        for (int index = 0; index < pathEnding.length; index++) {

                                                Node n =
document.selectSingleNode("/CHILDREN/CODE/CONCEPT_NAME/." + pathEnding[index]);

                                                if (n != null)
                                                {
                                                        n = n.getParent();
                                                        Document doc = MAMOUtils.getXmlFromText(n.asXML());

                                                        Text textValue = new Text(pathInit[index] + "." +
pathEnding[index] + "/" + doc.selectSingleNode("//CODE/VALUE/CODE_MEANING").getText());
                                                        valores.add( new
Text(createAliasForXPath(textValue.toString())) );
                                                }
                                        }

                                        compositeKey.setFila( valores );
                                        context.write(compositeKey, one/*new Text(valores.get(valores.size()-1))*/);
                                }
                        }
                        else
                        {
                                CompositeKey compositeKey = new CompositeKey();
                                compositeKey.setColumna(new Text("ERROR READING XML: " + key.toString() + "
["+xml+"]"));
                                compositeKey.setFila(new ArrayList<Text>());
                                context.write(compositeKey, one);
```

```
                }

        }


        @Override
        protected void cleanup(Mapper<Object, BytesWritable, CompositeKey, LongWritable/*Text*/>.Context
context) throws IOException, InterruptedException {

                heartBeat.stopbeating();
                super.cleanup(context);
        }


        private static String getDefaultCharSet() {
        OutputStreamWriter writer = new OutputStreamWriter(new ByteArrayOutputStream());
        String enc = writer.getEncoding();
        return enc;
    }

        private static String createAliasForXPath(String xpath)
        {
                String[] tokens = xpath.split("/");

                return new String("/" + tokens[2] + "/" + tokens[tokens.length-1]);
        }

}
```

9.3. Función de reducción

```
package test10.nivel2;

import java.io.IOException;

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;


public class TheReducer extends Reducer<CompositeKey, LongWritable/*Text*/,Text,LongWritable> {

        public void reduce(CompositeKey key, Iterable<LongWritable> values, Context context) throws IOException,
InterruptedException {
                long sum = 0;

                for (LongWritable value : values)
                {
                        sum += value.get();
                }

                context.write(new Text(key.toString()), new LongWritable(sum));
        }
}
```

9.4. Creación de clave compuesta para los mappers

```
package test10.nivel2;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
```

```
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.WritableComparable;

public class CompositeKey implements WritableComparable<CompositeKey> {

	private List<Text> filas;
	private Text columna;

	public CompositeKey() {
		this.filas = new ArrayList<Text>();
		this.columna = new Text();
	}

	public CompositeKey(List<Text> filas, Text columna) {

		this.filas = filas;
		this.columna = columna;
	}

	public List<Text> getFilas() {
		return this.filas;
	}

	public void setFila(List<Text> filas) {
		this.filas = filas;
	}

	public Text getColumna() {
		return columna;
	}

	public void setColumna(Text columna) {
		this.columna = columna;
	}

	@Override
	public String toString() {

		StringBuilder builder = new StringBuilder();

		builder.append(this.columna.toString()).append(",");

		for (Text fila : filas) {
			builder.append( fila.toString() );
			builder.append(",");
		}

		return builder.toString();
	}

	@Override
	public void readFields(DataInput in) throws IOException {

		int size = in.readInt();
		this.filas = new ArrayList<Text>(size);

		for(int i = 0; i < size; i++){
	    Text text = new Text();
	    text.readFields(in);
	    this.filas.add(text);
	  }

		this.columna.readFields(in);
	}

	@Override
	public void write(DataOutput out) throws IOException {

		out.writeInt(this.filas.size());

		for (Text l : this.filas) {
	    l.write(out);
	  }
```

```
                this.columna.write(out);
        }

        @Override
        public int compareTo(CompositeKey o) {

                int result = 0;

                result = this.columna.toString().compareTo( o.columna.toString() );

                String k1 = "";
                for (int i=0; i < this.filas.size(); i++) {
                        k1 = k1.concat(this.filas.get(i).toString());
                }

                String k2 = "";
                for (int i=0; i < o.getFilas().size(); i++) {
                        k2 = k2.concat(o.getFilas().get(i).toString());
                }

                if ( result == 0 )
                {
                        return k1.compareTo(k2);
                }

                return result;
        }


}
```

9.5. Creación de los keyComparator

```
package test10.nivel2;

import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.io.WritableComparator;

public class CompositeKeyComparator extends WritableComparator {

        protected CompositeKeyComparator() {
                super(CompositeKey.class, true);
        }

        @SuppressWarnings("rawtypes")
        @Override
        public int compare(WritableComparable w1, WritableComparable w2) {

                CompositeKey key1 = (CompositeKey) w1;
                CompositeKey key2 = (CompositeKey) w2;

                int compare = key1.getColumna().toString().compareTo(key2.getColumna().toString());

                if (compare == 0) {

                        String k1 = "";
                        for (int i=0; i < key1.getFilas().size(); i++) {
                                k1 = k1.concat(key1.getFilas().get(i).toString());
                        }

                        String k2 = "";
                        for (int i=0; i < key2.getFilas().size(); i++) {
                                k2 = k2.concat(key2.getFilas().get(i).toString());
                        }

                        return k1.compareTo(k2);
                }

                return compare;
        }
```

```
}
```

9.6. Creación de Partitioner

```
package test10.nivel2;

import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Partitioner;
import org.apache.hadoop.mapreduce.lib.partition.HashPartitioner;

public class ActualKeyPartitioner extends Partitioner<CompositeKey, Text> {

	HashPartitioner<Text, Text> hashPartitioner = new HashPartitioner<Text, Text>();
	Text newKey = new Text();

	@Override
	public int getPartition(CompositeKey arg0, Text arg1, int arg2) {

		if(arg2 == 0)
	return 0;

		try
		{
			newKey.set( arg0.getColumna() );
			return hashPartitioner.getPartition(newKey, arg1, arg2);
		}
		catch (Exception e)
		{
			e.printStackTrace();
			return (int) (Math.random() * arg2);
		}
	}

}
```

9.7. Creación de conocimiento (Gráfico con R)

```
install.packages("Rcpp")
install.packages("ggplot2")
install.packages("reshape2")

library(plyr)
library(reshape)
library(reshape2)
library(ggplot2)
library(grid)

fileInput <- args[1]

infodata <- read.csv(file=fileInput,sep=',',header=T, encoding = 'UTF-8')
names(infodata) <- c("BIRADS","TIPO_LESION","MARGEN","VALORES")
frame <- data.frame(infodata)
dataset_plyr <- ddply(frame, c("BIRADS","TIPO_LESION","MARGEN","VALORES"))

plot <- qplot(	main="Análisis de correlacción de factores",
	x=VALORES,
	y = TIPO_LESION,
	facets = BIRADS~MARGEN,
	data = dataset_plyr,
	alpha=I(0.9),
	size=I(2),
	shape = MARGEN,
	colour = BIRADS,
	geom="point",
```

```
            xlab="[ Rangos de Valores ]",
            ylab="[ Tipos de Lesión ]"
)

plot <- plot + theme_bw(base_size = 10, base_family = "")
plot <- plot + theme(plot.title = element_text(colour = "red", size = 40),
                  axis.text = element_text(colour = "black", face = "italic"),
                  strip.text.x = element_text(colour = "brown", angle = 0, size = 10, face = "italic"),
                  strip.text.y = element_text(colour = "blue", angle = 0, size = 10, face = "italic"),
                  axis.ticks.length = unit(.25, "cm"),
                  panel.margin = unit(.35, "cm"),
                  legend.background = element_rect(colour = "black"),
                  legend.background = element_rect(), legend.margin = unit(1, "cm"))
plot

ggsave(plot, file="image.pdf", scale=2)
```

9.8. Lanzamiento Cluster en EMR

```
aws emr create-cluster --name "MyClusterAlucloud95" --ami-
version 3.3 --applications Name=Hue \
--log-uri s3://trencadis/logs/ \
--use-default-roles --ec2-attributes KeyName=alucloud95-
keypair2 \
--instance-type m1.medium --instance-count 8 \
--steps Type=CUSTOM_JAR,Name="Java
App",ActionOnFailure=CONTINUE,Jar=s3://trencadis/xml/Testi
ngProject.jar,MainClass=test10.nivel2.MainTestToolRunner,\
Args=["s3n://trencadis/seqfile/","s3n://trencadis/exit2/",
"s3n://trencadis/xml/configuracion.properties"]
```