UNIVERSIDAD CENTRAL “MARTA ABREU” DE LAS VILLAS

FACULTAD DE MATEMÁTICA, FÍSICA Y COMPUTACIÓN

LICENCIATURA EN CIENCIA DE LA COMPUTACIÓN

Medición de la dimensión completitud en la base de datos MARC de ABCD

Trabajo de Diploma

Autor: Dariel León Hernández

Tutores: MSc. Lisandra Díaz de la Paz
Lic. Juan Luis García Mendoza

Santa Clara, Cuba, 2015

DICTAMEN

Hago constar que el presente trabajo fue realizado en la Universidad Central “Marta Abreu” de Las Villas como parte de la culminación de los estudios de la especialidad de Ciencia de la Computación, autorizando a que el mismo sea utilizado por la institución, para los fines que estime conveniente, tanto de forma parcial como total y que además no podrá ser presentado en eventos ni publicado sin la autorización de la Universidad.

Firma del autor

Los abajo firmantes, certificamos que el presente trabajo ha sido realizado según acuerdos de la dirección de nuestro centro y el mismo cumple con los requisitos que debe tener un trabajo de esta envergadura referido a la temática señalada.

Firma del tutor

Firma del jefe del Laboratorio

DEDICATORIA

A mi Dios, por haberme dado la oportunidad de llegar hasta aquí.

A mis padres, por sus grandes sacrificios en todos mis años de estudio, por ser mis guías e inspiración y por haber estado presentes en las buenas y en las malas.

A mi esposa por su apoyo en los momentos difíciles.

AGRADECIMIENTOS

A mi Dios, porque sin Él no hubiera llegado aquí.

A mis padres, por ser los mejores padres del mundo, por sus consejos y valores que me ayudaron a continuar, por brindarme su sabiduría.

Al Dr. Amed Abel Leiva Mederos, por la colaboración brindada sobre dominio de aplicación.

A mis tutores Lisandra y Juan Luis, por su ayuda invaluable.

A mi esposa por haberme soportado con ternura mis días de arduo trabajo.

A mis pastores por su apoyo incondicional y sus oraciones.

A mi equipo de estudio (Adrián, Batman, Charlie y Jimmy), por su apoyo y ayuda durante cinco largos años. ¡Al fin terminamos!

A todos los que me apoyaron.

“En la escuela se ha de aprender el manejo de las fuerzas con que en la vida se ha de luchar”.

José Martí

RESUMEN

En la actualidad ha tomado auge el desarrollo de tecnologías para gestionar *big data*. Estas constituyen un punto esencial en la extracción, almacenamiento, limpieza y análisis de la misma.

En las bibliotecas de la Universidad Central “Marta Abreu” de Las Villas (UCLV) se cuenta con un sistema para la administración de registros bibliográficos. Hasta el momento, los datos con los que se cuentan poseen disímiles errores y la no presencia de un *software* de perfilado de datos dificultaba la medición de su calidad. Por tanto, el presente trabajo de diploma tiene como objetivo implementar trabajos con el modelo de programación MapReduce para medir la dimensión completitud. Además, se crea una herramienta que permite la interacción entre la base de datos utilizada y la fuente de datos.

Para lograr estos objetivos se utiliza el marco de trabajo Hadoop para ejecutar los trabajos MapReduce, es decir, para realizar el procesamiento de los datos y la base de datos distribuida MongoDB para su almacenamiento.

Palabras claves: *Big data*, dimensión completitud, Hadoop, medir, MongoDB.

ABSTRACT

Today the development of technologies has boomed to arrange big data. These constitute an essential point in extraction, storing, cleaning and analysis of them.

At the Central University “Marta Abreu” de Las Villas libraries, there is a system in order to administer bibliographic records. So far, the data that are available, have dissimilar mistakes and not having a software to profile data make difficult the measurement of quality. Thus, this paper has an objective to implement jobs with the MapReduce programming model to measure the completeness dimension. Beside it is created a tool that allows interactions between the used database and the data source.

In order to achieve these objectives a framework named Hadoop is used to implement MapReduce jobs, that is, to carry out the processing of data and MongoDB distributed database for its storage.

Keywords: Big data, completeness dimension, Hadoop, measure, MongoDB.

TABLA DE CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

INTRODUCCIÓN

El continuo incremento en el volumen y detalle de los datos capturados (en el rango de los exabytes, es decir, 10^{18} bytes y superior) por organizaciones como los medios sociales, internet de las cosas (*Iot*, *Internet of things*), multimedia, entre otras, han producido un abrumador flujo de datos en diversos formatos: estructurados, semiestructurados y no estructurados (Hashem et al., 2014). Eric Schmidt, director de Google desde el año 2001 hasta el 2011, expresó que debe tenerse en cuenta que "desde el amanecer de la civilización hasta el 2003, se crearon más de cinco exabytes de información y en la actualidad, esta cantidad se está generando cada dos días", es decir, aproximadamente 2.5 exabytes de datos diariamente.

El volumen es sólo uno de los aspectos de *big data*; otros atributos son variedad y velocidad. La creación de los datos está ocurriendo a una proporción gigantesca, refiriéndose aquí como *big data* y surgiendo como una tendencia ampliamente reconocida, caracterizándose, según (Hashem et al., 2014), por tres aspectos:

- Gran cantidad de datos.
- Los datos no pueden ser categorizados dentro de las bases de datos relacionales.
- Los datos son generados, capturados y procesados rápidamente.

El actual crecimiento en la cantidad de datos coleccionados es asombrosa, ver Tabla 1. Según (Stephen Kaisler et al., 2013), un mayor desafío para los investigadores y profesionales en esta rama es que este crecimiento es rápido y está excediendo las habilidades para:

- Diseñar sistemas apropiados para manejar los datos de manera efectiva.
- Analizar los datos para extraer significados relevantes para la toma de decisiones.

Conjunto de datos	**Descripción**
Large Hadron Collider/Particle Physics (CERN [1])	13-15 petabytes in 2010
Internet Communications	667 exabytes in 2013
Social Media	Más de 12 terabytes se tweets cada día
British Library UK Website Crawl	Aproximadamente 110 terabytes por dominio

Tabla 1: Algunos ejemplos de big data. (Stephen Kaisler et al., 2013)

[1] En español, Consejo Europeo para la Investigación Nuclear.

Como se puede observar, estos inmensos volúmenes de datos no pueden ser procesados eficientemente por las herramientas y métodos tradicionales de las bases de datos, por lo que ha propiciado la creación de varios modelos de programación para el procesamiento de datos que intentan ofrecer un acercamiento a una solución para *big data*.

En las bibliotecas de la UCLV se implantó en el 2011 el sistema de Automatización de Bibliotecas y Centros de Documentación (ABCD). Dentro de la cual se encuentra la base de datos NoSQL ISIS orientada a documentos denominada MARC que almacena datos semiestructurados en el formato estandarizado MARC21, de la cual se extraen los archivos *iso* que se utilizan como fuentes de datos. Como los mismos fueron introducidos de forma manual acarreó como consecuencia la existencia de problemas de calidad de datos tales como: pérdida de información, campos incompletos, valores ausentes, datos escritos incorrectamente, no estandarización de valores, entre otros.

Objetivo General: Evaluar la dimensión completitud en los registros bibliográficos del módulo Catalogación del sistema de Automatización de Bibliotecas y Centros de Documentación (ABCD) implantado en las bibliotecas de la UCLV, utilizando tecnologías *big data*.

Objetivos Específicos:

1. Analizar las principales características y funcionalidades de las herramientas que trabajan con *big data*.
2. Configurar una máquina virtual con la instalación de Hadoop y MongoDB.
3. Implementar una herramienta para la importación, exportación de archivos iso hacia MongoDB y permita además su visualización parcial.
4. Implementar una métrica para la evaluación de la dimensión de calidad de los datos completitud en la base de datos NoSQL denominada MARC perteneciente al módulo Catalogación del sistema ABCD, utilizando Hadoop/MapReduce.

Preguntas de investigación

1. ¿Cuáles son las principales características y funcionalidades de las herramientas que trabajan con *big data*?

2. ¿Qué pasos se deben seguir para configurar una máquina virtual donde se instalen las herramientas de *big data* Hadoop y la base de datos NoSQL MongoBD?

3. ¿Cómo implementar una herramienta para la importación, exportación de archivos iso hacia MongoDB y permita además su visualización parcial?

4. ¿Cómo implementar una métrica para la evaluación de la dimensión de calidad de los datos completitud en la base de datos NoSQL denominada MARC perteneciente al módulo Catalogación del sistema ABCD, utilizando Hadoop/MapReduce?

Justificación de la investigación

La investigación posee utilidad práctica porque diariamente se accede al sistema ABCD en donde se encuentran almacenados estos datos que contienen errores. Hasta el momento no se cuenta con una herramienta para el perfilado de los datos que tuviera como entrada la base de datos ISIS, lo cual dificulta la realización de dicho proceso, por lo que es necesario la implementación de métricas de calidad de datos utilizando las potencialidades de las tecnologías que *big data* ofrece.

Viabilidad

Para llevar a cabo la presente investigación se utiliza el marco de trabajo Hadoop para implementar una métrica de la dimensión completitud. Sobre esta herramienta existen trabajos previos los cuales aseveran su fiabilidad.

En el grupo de Base de Datos del Centro de Estudios Informáticos se cuenta con los medios técnicos, software y personal necesario para desarrollar este trabajo.

El presente trabajo se estructura en tres capítulos:

En el capítulo 1 se exponen las principales características de las herramientas para gestionar *big data* y su relación con la calidad de datos, en el capítulo 2 se presenta la configuración

del marco de trabajo Hadoop y la base de datos distribuida MongoDB y por último, en el capítulo 3, se muestra la implementación de una herramienta y dos trabajos MapReduce para la medición de una métrica de la dimensión completitud.

CAPÍTULO 1: PRINCIPALES CARACTERÍSTICAS DE LAS HERRAMIENTAS PARA LA GESTIÓN DE *BIG DATA*.

Este capítulo proporciona una visión general sobre *big data* y las herramientas relacionadas con el mismo; se muestran sus diversas definiciones, se aborda en sus características, la situación en cuanto a la calidad que poseen sus datos y los pasos que se deben seguir en el momento de procesar los grandes volúmenes de información presentes en *big data*. Las tendencias actuales de las tecnologías usadas en *big data* es otro de los aspectos que se tratan en este capítulo tales como: bases de datos NoSQL, Hadoop y sus bibliotecas, y MapReduce.

1.1 Definición de big data

El gran fenómeno que ha capturado la atención de la industria de la informática moderna hoy en día desde la Internet es *big data*. Años antes de que las personas en el mercado adoptaran el término *big data*, Doug Laney (analista del antiguo META Group, ahora denominado Gartner) publicó en el informe (2001) por primera vez la definición de la misma a través de tres V: volumen, velocidad y variedad.

La naturaleza de *big data* es indistinta e involucra procesos considerables para identificar y traducir los datos (Hashem et al., 2014). Varios investigadores y profesionales han utilizado los términos anteriormente mencionados. Por ejemplo, (Cox and Ellsworth, 1997) se refiere a *big data* como un volumen grande de datos científicos para la visualización. En (Manyika et al., 2011) se define *big data* como "la cantidad de datos que exceden la capacidad de la tecnología para almacenarlos, gestionarlos y procesarlos eficazmente". Entretanto, (Zikopoulos et al., 2012) y (Berman, 2013) definen *big data* de acuerdo a las tres V definidas por Laney.

- **Volumen:** se refiere a la cantidad de todos los tipos de datos generados de fuentes diferentes y en continua expansión. El beneficio de recoger cantidades grandes de datos incluye la creación de patrones ocultos a través del análisis de los mismos. (Hashem et al., 2014)
- **Variedad:** se refiere a los diferentes tipos de datos coleccionados a través de los sensores, *smartphones* o las redes sociales. Tales tipos de los datos incluyen video, imagen, texto, audio, ficheros *logs*, entre otros, presentándose en formato estructurado o no estructurado. La mayoría de los datos generado por aplicaciones

móviles está en el formato no estructurado. Por ejemplo, los mensajes del texto, juegos en línea, blogs y los medios de comunicación sociales generan tipos diferentes de datos no estructurado a través de los dispositivos móviles y sensores. Los usuarios de Internet también generan un conjunto sumamente diverso de datos estructurados, semiestructurados y no estructurado. (O'Leary, 2013)

- **Velocidad:** se refiere a la velocidad de transferencia de los datos. El contenido de los datos cambian constantemente debido a la absorción de colecciones complementarias de datos, la introducción de datos previamente archivados o colecciones heredadas y flujos de datos que llegan de múltiples fuentes (Berman, 2013).

Otros autores agregan otras V como retos o desafíos para *big data*. Como por ejemplo (Gantz and Reinsel, 2011) y (Chen et al., 2014) especifican una cuarta V denominada valor.

- **Valor:** es el aspecto más importante de *big data*; se refiere al proceso de descubrir gran cantidad de valores ocultos en conjuntos de datos muy grande que contienen varios tipos y se generan rápidamente. (Chen et al., 2014)

En (Ebner et al., 2014) y (Syed et al., 2013) sustituye la valor por veracidad y define *big data* como "un fenómeno caracterizado por un continuo aumento en el volumen, variedad, velocidad y veracidad de datos que exigen de técnicas y tecnologías avanzadas para capturar, almacenar, distribuir, gestionar y analizar estos datos" (Ebner et al., 2014).

En (Moreno, 2014) y (Demchenko et al., 2013) define como reto a valor y le agrega como segundo desafío la veracidad (en el presente trabajo se toma valor y veracidad como el cuarta y quinta V respectivamente), definiéndola como :

- **Veracidad:** se refiere al nivel de fiabilidad asociado a ciertos tipos de datos. Esforzarse por conseguir unos datos de alta calidad es un requisito importante y un reto fundamental de *big data*, pero incluso los mejores métodos de limpieza de datos no pueden eliminar la imprevisibilidad inherente de algunos datos, como el tiempo, la economía o las futuras decisiones de compra de un cliente. La necesidad de reconocer y planificar la incertidumbre es una dimensión de *big data* que surge a

medida que los directivos intentan comprender mejor el mundo incierto que les rodea. (Moreno, 2014)

En el artículo (Rijmenam, 2015) se agregan además dos retos, las cuales son:

- **Variabilidad:** *big data* es sumamente variable. Brian Hopkins, un analista principal de *Forrester*, define la variabilidad como la "variación en el significado, en el léxico". Además, se refirió a la supercomputadora Watson que ganó el juego *Jeopardy*, como el mayor ejemplo para ilustrarlo. La supercomputadora tenía que "analizar una respuesta en sus variados significado y […] deducir cual era la pregunta correcta". Eso es sumamente difícil porque las palabras tienen significados diferentes dependiendo del contexto. Para la respuesta correcta, Watson tenía que entender el contexto.

  De esta manera, la variabilidad es muy relevante al realizar análisis de opiniones. La variabilidad se refiere al hecho de que el significado está cambiando (rápidamente). En casi todos los mismos *tweets*, una palabra puede tener un significado totalmente diferente. Para realizar un análisis de opiniones apropiado, los algoritmos necesitan poder entender el contexto y poder descifrar el significado exacto de una palabra en ese contexto. Esto todavía es muy difícil.
- **Visualización:** Ésta es la parte más difícil de *big data*. Hacer que toda esa inmensa cantidad de datos sea comprensible, de manera que sea fácil de entender y leer. Con los análisis y visualizaciones correctas pueden usarse los datos de baja calidad, de otra manera serían inútiles. Las visualizaciones no significan gráficos ordinarios o mapas de pastel. Se refiere a gráficos complejos que pueden incluir muchas variables de datos mientras que todavía siga permaneciendo entendibles y legibles.

  Visualizar no es necesariamente la parte tecnológica más difícil sino la más desafiante. Contar una historia compleja en un gráfico es una ardua tarea pero también sumamente crucial.

Según (Hashem et al., 2014) , *big data* es un conjunto de técnicas y tecnologías que requieren nuevas formas de integración para descubrir valores ocultos en conjuntos de datos grandes, diversos, complejos y de una escala masiva.

En definitiva, *big data* (ver Figura 1) es una combinación de estas características que crea

una oportunidad para que las empresas puedan obtener una ventaja competitiva en el actual mercado digitalizado (Schroeck et al., 2012). Permite a las empresas transformar la forma en que interactúan con sus clientes y les presentan servicios. No todas las organizaciones adoptarán el mismo enfoque con respecto al desarrollo y la creación de sus capacidades de *big data.* Sin embargo, en todos los sectores existe la posibilidad de usar las nuevas tecnologías y potencialidades analíticas que ofrece *big data* para mejorar la toma de decisiones y el rendimiento. Estos factores hacen que *big data* sea difícil de capturar, extraer y gestionar usando métodos tradicionales. (Moreno, 2014)

Figura 1: Definición y desafíos de big data.

1.2 Principales características

De acuerdo con (Hashem et al., 2014), *big data* se clasifica en diferentes categorías para entender mejor sus características. La Figura 2 muestra las numerosas categorías de *big data*. La clasificación es basada en cinco aspectos: las fuentes de los datos, el formato del contenido, el almacenamiento de los datos, representación de los datos y el procesamiento de los datos.

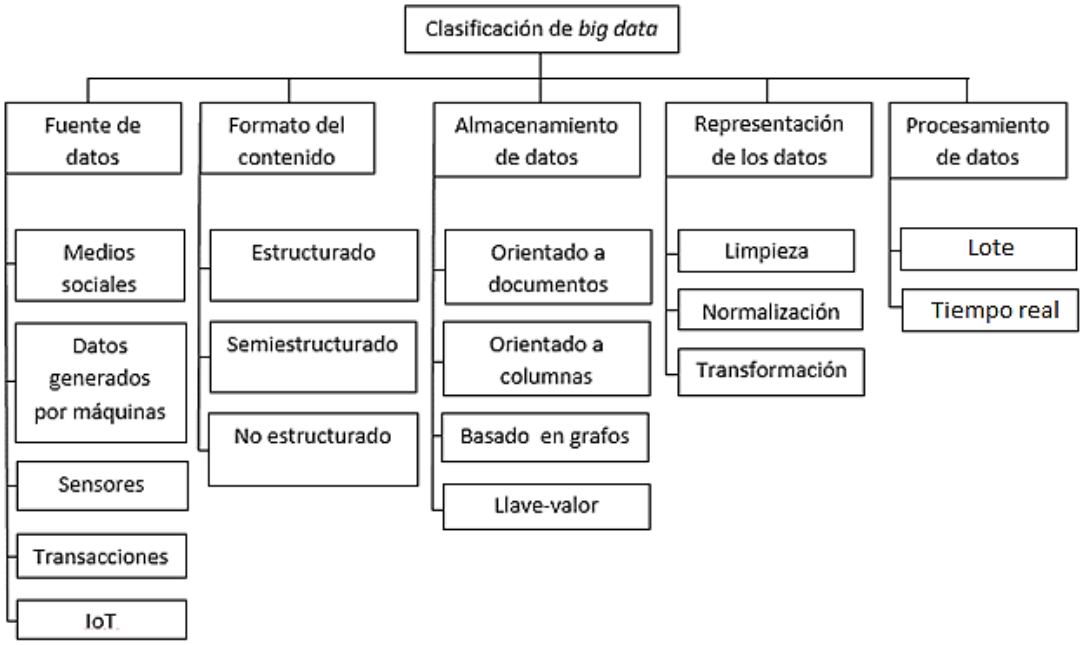

Figura 2: Clasificación de big data. (Hashem et al., 2014)

Cada una de estas categorías tiene sus propias características y complejidades como se observa en la Tabla 2. Las fuentes de datos incluyen internet de las cosas, sensores y datos transaccionales, desde estructurados y semiestructurados hasta no estructurados. Las más populares son las bases de datos relacionales de las cuales existe una gran variedad (Nugent et al., 2013). Como resultado de la gran variedad de fuentes, los datos capturados difieren en el tamaño respecto a la redundancia, consistencia, ruido, entre otras (Hashem et al., 2014).

Clasificación		**Descripción**
Fuente de datos	Medios sociales	Los medios sociales son la fuente de información generada vía URL para compartir o intercambiar información e ideas en comunidades y redes virtuales, tales como proyectos colaborativos, *blogs*, *microblogs*, Facebook y Twitter.
	Datos generados por máquinas	Los datos generados por máquinas son datos que se generan automáticamente de un hardware o software como computadoras, dispositivos médicos u otras máquinas sin intervención humana.
	Sensores	Varios dispositivos sensoriales existen para medir las variables físicas y cambiarlos en señales.
	Transacciones	Los datos de transacciones financieras y de trabajo, abarcan un evento que involucra una dimensión de tiempo para describir los datos.
	IoT	IoT representa un conjunto de objetos que son

		únicamente identificables como una parte de la Internet. Estos objetos incluyen *smartphones*, cámaras digitales, y *tablets*. Cuando estos dispositivos se conectan entre sí a través de Internet, habilitan más procesos inteligentes y servicios que soportes básicos, económicos, medioambientales y necesidades de la salud. Un gran número de dispositivos conectado a Internet proporciona muchos tipos de servicios y produce cantidades grandes de datos e información (Rao et al., 2012).
Formato del contenido	Estructurados	Los datos estructurados se manejan a menudo con SQL. Son fáciles de introducir, consultar, almacenar y analizar. Los ejemplos de datos estructurados incluyen números, palabras y fechas.
	Semiestructurados	Los datos semiestructurados no siguen un sistema de base de datos convencional. Pueden estar en forma de datos estructurados que no están organizados en los modelos de las base de datos relacionales, como las tablas. Capturar datos semiestructurados para analizar es diferente de capturar un formato de archivo fijo. Por consiguiente, capturar los datos semiestructurados requiere el uso de reglas complejas que deciden dinámicamente el próximo proceso después de capturar los datos (Franks, 2012).
	No estructurados	Los datos no estructurados, como los mensajes de texto, información de localización, videos y los datos de los medios de comunicación sociales, son datos que no siguen un formato específico. Considerando que el tamaño de este tipo de dato continúa aumentando a través del uso de *smartphones*, la necesidad de analizar y entenderlos se ha vuelto un desafío.
Almacenamiento de datos	Orientado a documentos	El almacenamiento de datos orientado a documentos esta principalmente diseñado para almacenar y recuperar colecciones de documentos o información y soporta formas complejas de los datos en varios formatos estándares, como JSON, XML y archivos binarios (ej., PDF y Microsoft Word). El almacenamiento de datos orientado a documentos es similar a un registro o fila en una base de datos relacional pero es más flexible y puede recuperar documentos basados en sus contenidos (ej., MongoDB y CouchDB).
	Orientado a columnas	Una base de datos orientada a columnas guarda su contenido en las columnas aparte de las filas, con los valores de los atributo perteneciendo a la misma columna contigua almacenada. Orientado a columnas es diferente a los sistemas de la base de datos clásicos que almacenan filas enteras una después de la otra (Abadi et al., 2009), como BigTable (Du et al., 2014).
	Basadas en grafos	Una base de datos basada en grafos, como Neo4j, está diseñada para almacenar y representar datos utilizando un modelo de grafo con nodos, aristas y propiedades conectadas entre sí a través de las relaciones (Neubauer,

		2010).
	Llave-valor	Llave-valor es un sistema de base de datos relacional alternativo que almacena y accede a los datos para ser escalables y de un gran tamaño (Seeger and Ultra-Large-Sites, 2009). Dynamo (DeCandia et al., 2007) es un ejemplo bueno de un sistema de almacenamiento llave-valor ampliamente disponible. Es usado por amazon.com en algunos de sus servicios. Otros ejemplos de almacenamiento llave-valor son Apache Hbase (Taylor, 2010), Apache Cassandra (Lakshman and Malik, 2011), y Voldemort. Hbase usa HDFS, una versión de software libre del BigTable de Google construida en Hadoop. Hbase almacena datos en tablas, filas y celdas. Las filas se ordenan por una fila llave. Cada celda en una tabla es especificada por una fila llave, una llave columna y una versión, con el valor contenido como un arreglo de bytes no interpretable.
Representación de los datos	Limpieza	"La limpieza de datos es el proceso mediante el cual se realizan transformaciones a los datos para que adquieran la calidad que les permita aportar información confiable, a fin de contribuir a la eficiencia y efectividad de los procesos de toma de decisiones" (Porrero, 2011).
	Transformación	Transformación es el proceso de convertir los datos en una forma apta para su análisis. (Hashem et al., 2014)
	Normalización	Normalización es el método de estructurar los esquemas de las bases de datos para minimizar la redundancia (Quackenbush, 2002).
Procesamiento de datos	Lote	Los sistemas basados en MapReduce han sido adoptados por muchas organizaciones en los últimos años para los trabajos de grandes lotes (Chen et al., 2012). Tales sistemas permiten el escalado de aplicaciones a través de un gran clúster de máquinas de miles de nodos.
	Tiempo real	Una de las más famosas y poderosas herramientas de procesamiento en tiempo real de *big data* es S4 (Neumeyer et al., 2010). S4 es una plataforma de computación distribuida que les permite a los programadores desarrollar aplicaciones para el procesamiento continuo de flujo ilimitado de datos. S4 es una plataforma escalable, parcialmente tolerante al fallo, de propósito general y adaptable a varias plataformas.

Tabla 2: Descripción de la clasificación de Big Data (Hashem et al., 2014).

1.3 Calidad de datos

Es difícil proporcionar una definición universal del significado de calidad. Según (Bobrowski et al., 1999), cuando se habla sobre la calidad no siempre se refiriere al mismo concepto. Por ejemplo, "suponer que se está planeando un viaje a un país extranjero. Se tiene que escoger una aerolínea para volar. ¿Cuál es la mejor? Claro, se preferirá la

aerolínea que ofrece la mejor calidad. ¿Pero, qué significa calidad? Se quiere llegar a tiempo, los asientos más cómodos, una tripulación útil, un viaje callado y los precios más bajos. Estos atributos (puntualidad, consuelo, utilidad, paz y precios bajos) constituyen la noción de calidad en este contexto en particular. Incluso en la situación del viaje, alguien más no se preocupa por el precio, pero está muy angustiado con las comidas servidas. Así que su noción de calidad de la aerolínea es diferente al primero. Puede diferir no solamente en los atributos tenidos en cuenta; la relevancia de cada artículo puede ser diferente. Es más, se puede tener nociones diferentes de calidad de la aerolínea para viajes diferentes" (Bobrowski et al., 1999).

Uno de los conceptos más referenciados para el término de calidad de datos fue propuesto en 1980 por Joseph Juran y Frank Gryna en el libro titulado "*Quality planning and analysis*". En (Wang and Strong, 1996) se cita como "*data that are fit for use by data consumer*" o datos adecuados para el uso del consumidor de datos. Lo que expresa que el consumidor es el encargado de valorar el nivel de calidad de un conjunto de datos usados para una determinada tarea, la cual está elaborada en un contexto específico (Strong et al., 1997).

Según el estudio sobre las dimensiones de calidad de datos realizado en (Wang and Strong, 1996), se crearon cuatro categorías para agruparlas. Por ejemplo, se le llamó calidad de datos intrínseca a la categoría que agrupa las dimensiones exactitud, objetividad, credibilidad y reputación. En la Figura 3 se presenta un árbol que engloba las categorías con sus respectivas dimensiones de calidad de datos. En la Tabla 3 se describe una muestra de cuatro dimensiones de calidad de datos.

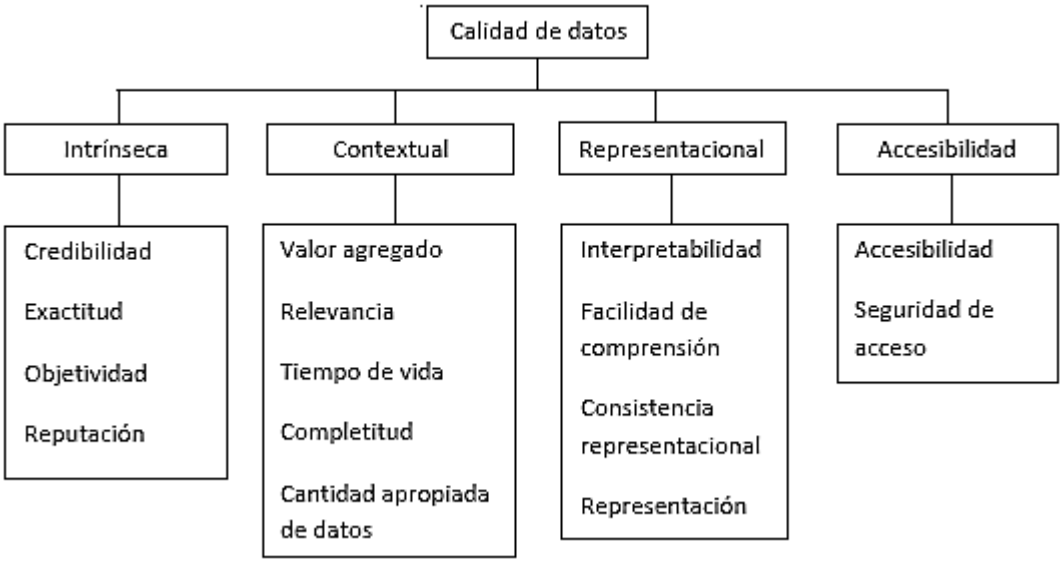

Figura 3: Marco de trabajo conceptual de la calidad de datos. (Wang and Strong, 1996)

Dimensión	**Descripción**	**Referencia**
Exactitud	Grado en que los datos son correctos, confiables y certificados.	(Wang and Strong, 1996)
Completitud	Hasta qué punto los datos no están perdidos y cubren las necesidades de las tareas y son de anchura y profundidad suficientes para la tarea a mano.	(Pipino et al., 2002)
Consistencia	Violación de reglas semánticas definidas sobre un conjunto de datos.	(Batini et al., 2009)
Tiempo de vida	Grado en el que la edad de los datos es apropiada para la tarea en cuestión.	(Wang and Strong, 1996)

Tabla 3: Descripción de las dimensiones.

1.3.1 Big data y calidad de datos

Estudios recientes han mostrado que prevalece la calidad pobre en los datos de las grandes bases de datos y en la Web. La pobre calidad en los datos puede tener serias consecuencias en los resultados al analizarlos, la importancia de la veracidad, una de las V de *big data* está reconociéndose cada vez más. Debido al volumen y velocidad de los datos, se necesita entender y, posiblemente, limpiar los datos erróneos de manera escalable y veloz. Con la variedad de datos, provenientes de una diversidad de fuentes, las reglas de calidad de datos no pueden ser especificadas a priori; se necesita permitir que los datos "hablen por sí mismos" para descubrir su semántica. (Saha and Srivastava, 2014)

Con el gran volumen de datos generados, la rápida velocidad con que llegan y su gran variedad heterogénea, la calidad de los datos está lejos ser perfecta. Se ha estimado que los

datos erróneos traen consigo pérdidas económicas; por ejemplo: en los negocios de EE.UU, se pierden 600 mil millones dólares anualmente por causa de problemas de calidad en los datos (Eckerson, 2002). Según (Fan and Geerts, 2012) y (Redman, 1998) las empresas normalmente encuentran errores en los datos en una proporción de aproximadamente 1 al 5%, y para algunas compañías, es alrededor del 30%. En la mayoría de los proyectos de almacenes de datos, la limpieza de los datos cuesta del 30 al 80% del tiempo de desarrollo y consume más presupuesto mejorar la calidad de los datos que construir el sistema. (Saha and Srivastava, 2014)

Con el advenimiento de *big data*, la dirección de calidad de los datos se ha vuelto más importante que nunca. Típicamente, volumen, velocidad y variedad se usan para caracterizar las propiedades importantes de *big data*, pero extraer el valor y hacer que *big data* sea operacional, incrementa la importancia de la quinta "V" de *big data*, veracidad. La veracidad se refiere directamente a la inconsistencia y problemas de calidad de datos. Como (Tee, 2013) afirma, uno de los problemas más grandes con *big data* es la tendencia de los errores a incrementarse como las bolas de nieve. Los errores de entrada de un usuario, redundancia y corrupción afectan el valor de datos. Según (Sarsfield, 2011) y (Saha and Srivastava, 2014), sin la apropiada gestión de la calidad de los datos, incluso los errores menores pueden aumentar resultando en la pérdida de ingresos, ineficiencia del proceso y fracaso al no cumplir con las regulaciones de la industria y el gobierno.

1.4 Fases para el manejo de big data

El análisis de *big data* implica múltiples fases distintas donde cada una presenta grandes dificultades como la heterogeneidad, tiempo de vida y privacidad de los datos, además de la escalabilidad y la colaboración humana que influye de manera significativa al aumento del error en los datos. Centrarse solamente en la fase de análisis/modelado no es recomendable porque, aunque esta fase es crucial, es de poca utilidad sin las otras fases de la tubería de análisis de datos (ver Figura 4), de hecho, grandes retos se extienden más allá de dicha fase (Agrawal et al., 2012).

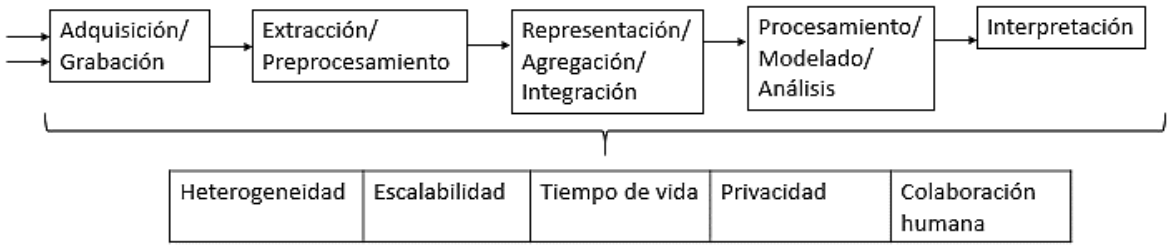

Figura 4: Tubería de análisis big data. (Agrawal et al., 2012)

Según (Agrawal et al., 2012) las etapas son:

1. Adquisición y grabación de datos:

   Big data se nutre de muchas y variadas fuentes de generación de datos. Por ejemplo, los experimentos científicos y simulaciones pueden producir fácilmente petabytes de datos.

   Muchos de los datos que se generan actualmente no son de interés, y se pueden filtrar y comprimir. Uno de los retos es definir estos filtros de tal manera que no descarten información útil. Es necesario realizar un procesamiento inteligente en los datos en bruto y reducirlos a un tamaño que los usuarios puedan manejar sin que exista ausencia de datos útiles. Por otra parte, se requieren técnicas de análisis *on-line* que pueden procesar dichos flujos de datos sobre la marcha, ya que es más difícil almacenarlos primero y reducirlos después.

2. Extracción y preprocesamiento de la información:

   Con frecuencia, la información recogida no está en un formato listo para su análisis. Por ejemplo, al considerar la recopilación de registros electrónicos de salud en un hospital que comprende dictados transcritos de varios médicos, datos estructurados de sensores y mediciones (posiblemente con cierta incertidumbre asociada) y datos de imagen como radiografías. No se puede dejar los datos en estos formatos y todavía realizar un análisis con eficacia. Más bien se requiere un proceso de extracción de información necesaria de las fuentes subyacentes y expresarlo en un formato adecuado para su análisis. Realizar esto correctamente y por completo es un desafío técnico, si se tiene en cuenta que estos datos también incluye imágenes y tal vez videos. Dicha extracción es a menudo altamente dependiente de la aplicación (por ejemplo, lo que quiere procesarse de una resonancia magnética es muy

diferente de lo que se puede procesar de una imagen de las estrellas o una foto de vigilancia).

3. Representación, agregación e integración de datos:

   Dada la heterogeneidad de esta gran cantidad de datos, no es suficiente con solo almacenarlo en un repositorio. Si sólo tenemos muchos conjuntos de datos en un repositorio, es poco probable que alguien alguna vez sea capaz de encontrarlos, y mucho menos reutilizar estos datos. Aún con los metadatos adecuados los desafíos se mantendrán debido a diferencias en los detalles experimentales y en la estructura del registro de los datos.

   El análisis de datos es mucho más difícil que simplemente localizar, identificar, entender y citar los datos. Para realizar un análisis eficaz a gran escala se debe ejecutar de manera completamente automatizada. Esto requiere diferencias en la estructura de datos y la semántica para ser expresados en formas comprensibles por el ordenador. Existen fuertes herramientas que trabajan con la integración de datos que puede proporcionar ciertas facilidades. Sin embargo, se requiere de un considerable esfuerzo para lograr este objetivo.

   Incluso para los análisis más simples que dependen de un único conjunto de datos, sigue existiendo una importante cuestión del diseño adecuado de la base de datos. Por lo general, hay muchas alternativas al almacenar la misma información. Ciertos diseños tendrán ventajas para ciertos fines y posiblemente inconvenientes para otros. Por ejemplo, la gran variedad en la estructura de las bases de datos de bioinformática con información sobre entidades sustancialmente similares, tales como genes.

4. Procesamiento, modelado y análisis de datos:

   Los métodos para la consulta y la minería de *big data* son diferentes al análisis estadístico tradicional en muestras pequeñas. *Big data* es a menudo ruidosa, dinámica, heterogénea y poco fiable. Sin embargo, incluso ruidosa podría ser más valioso que pequeñas muestras. Además, *big data* forma grandes redes de información heterogénea, con la que la información de redundancia puede ser explorada para compensar la falta de datos, para cotejar los casos conflictivos, para

validar las relaciones de confianza, de revelar agrupaciones inherentes y para descubrir relaciones y modelos ocultos.

La minería de datos también se puede utilizar para ayudar a mejorar la calidad y confiabilidad de los datos, comprender su semántica y proporcionar funciones y consultas inteligentes. Como se señaló anteriormente, los registros médicos de la vida real tienen errores, son heterogéneos y con frecuencia se distribuyen a través de múltiples sistemas. El valor del análisis de *big data* en el cuidado de la salud, para tomar sólo un ejemplo del dominio de aplicación, sólo puede realizarse si se puede aplicar con firmeza en estas condiciones difíciles. Por otro lado, el conocimiento desarrollado a partir de los datos puede ayudar en la corrección de errores y la eliminación de la ambigüedad.

5. Interpretación de los datos:

   Tener la capacidad de analizar grandes volúmenes de datos tiene un valor limitado si los usuarios no pueden entender el análisis. Por lo general, se trata de examinar todas las suposiciones hechas y volver sobre el análisis. Además, como se ha visto anteriormente, existen muchas fuentes posibles de error: los sistemas informáticos pueden tener errores, los modelos casi siempre tienen supuestos y los resultados pueden ser basado en datos erróneos. Por todas estas razones, el usuario responsable no cederá la autoridad al sistema informático. Más bien se trata de comprender y verificar los resultados producidos. El hecho de que un sistema informático haga más fácil esta interpretación es particularmente un reto en *big data* debido a su complejidad. A menudo hay supuestos cruciales detrás de los datos almacenados. En pocas palabras, es rara la vez que se proporcionan solo los resultados. Más bien, se debe proporcionar información adicional que explica cómo se derivó cada resultado y basada precisamente en la entrada de los datos. Dicha información complementaria es llamada procedencia de los datos resultantes. Al estudiar la mejor manera de capturar, almacenar y consultar la procedencia de los mismos, junto con técnicas adecuadas para capturar metadatos, se puede crear una infraestructura para proporcionar a los usuarios la capacidad tanto de interpretar los resultados analíticos obtenidos como repetir el análisis con diferentes supuestos, parámetros o conjuntos de datos.

1.5 Tecnologías relacionadas con big data

Las compañías alrededor del mundo ya están llevando a cabo sus planes para gestionar una solución *big data*. En cuanto a las tecnologías, *big data* presenta ciertas tendencias, como por ejemplo el marco de trabajo Hadoop. Esta herramienta está siendo usada por Microsoft, eBay, Facebook, IBM, LinkedIn, The New York Times, Twitter, entre otras compañías. En la Figura 5 se pueden apreciar algunas de estas tendencias.

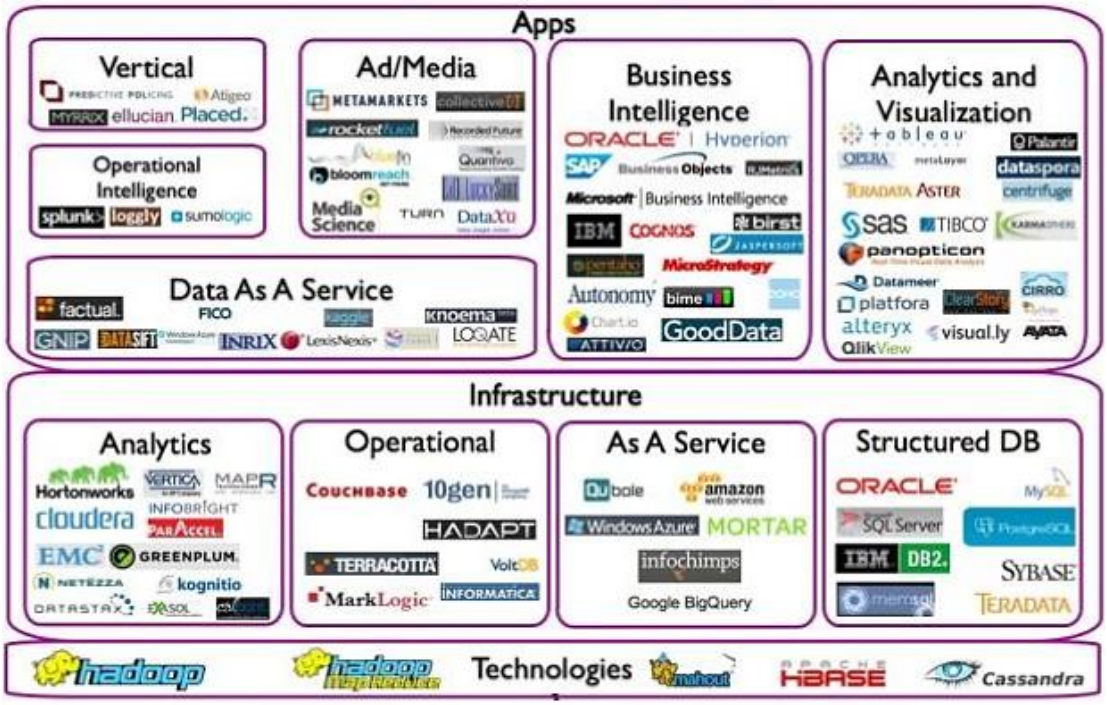

Figura 5: Tendencias de big data. (Syed et al., 2013)

Como resultado del gran desarrollo de *big data*, un nuevo tipo de tecnología ha surgido y está siendo utilizado en disímiles análisis de datos con estas características. Las tecnologías relacionadas con el análisis de datos incluyen, bases de colecciones enormes de datos NoSQL, Hadoop y MapReduce. Estas tecnologías forman el núcleo de un marco de software de código abierto que soporta el procesamiento de grandes volúmenes de datos a través de un clúster de computadoras.

En la actualidad existen diferentes herramientas de software para el tratamiento de la tecnología *big data*. Los *softwares* de tratamiento de grandes almacenes de datos tienen su base de programación en MapReduce. Entre ellos, Hadoop es uno de los más utilizados.

1.5.1 NoSQL (Not only SQL)

Muchas organizaciones reúnen inmensas cantidades de clientes, científicos, ventas y otros datos para un análisis futuro. Tradicionalmente, la mayoría de estas organizaciones ha guardado los datos estructurados en las bases de datos relacionales para el posterior acceso y análisis. En la era de *big data* existen más formatos en el contenido de los datos: semiestructurados y no estructurados, como se mencionó anteriormente, pero las bases de datos relacionales no son capaces de gestionarlas adecuadamente. Por tanto tienen una serie de limitaciones, como se ilustra en la Tabla 4:

Aspecto	**Descripción**
Escalabilidad	Los usuarios pueden mejorar el rendimiento y almacenamiento de una base de datos relacional ejecutándolo en una computadora más poderosa y cara. Sin embargo, para ampliarse más allá de un cierto punto debe distribuirse por múltiples servidores. Las bases de datos relacionales no trabajan fácilmente en forma distribuida porque unir las tablas a través sistema distribuido es difícil. Además, las bases de datos relacionales no se diseñan para funcionar con datos particionados, por lo que distribuir sus funcionalidades es una tarea muy complicada.
Complejidad	Con las bases de datos relacionales, los usuarios deben convertir todos los datos en tablas. Cuando los datos no encajan fácilmente en una tabla, la estructura de la base de datos puede ser compleja, difícil y lenta para trabajar.
Lenguaje SQL	Usar SQL es conveniente con los datos estructurados. Sin embargo, usando este lenguaje con otros tipos de información es difícil porque se diseña para trabajar con bases de datos relacionales, estructuradas con tablas fijas. SQL puede implicar grandes cantidades de código complejo y no trabaja bien con el desarrollo moderno y ágil.
Conjuntos con muchas características	Las bases de datos relacionales ofrecen una gran cantidad de característica en los conjuntos e integridad en los datos, pero los defensores de las NoSQL refieren que a menudo los usuarios de la base de datos no necesitan todas las características que ofrecen, tanto como el costo y complejidad que ellos agregan.

Tabla 4: Limitaciones de las bases de datos relacionales respecto a big data (Leavitt, 2010).

Por otro lado, un número creciente de diseñadores y usuarios han empezado a volverse a varios tipos de bases de datos no relacionales, llamadas NoSQL. Una base de datos NoSQL provee mecanismos para almacenar y recuperar grandes volúmenes de datos distribuidos. Las bases de datos no relacionales, incluyendo bases de datos jerárquicas, de grafo y orientadas a objeto, han existido desde los finales de 1960 (Leavitt, 2010).

Sin embargo, están desarrollándose nuevos tipos de bases de datos NoSQL y sólo ahora es que empiezan a ganar popularidad en el mercado. Las bases de datos NoSQL toman diferentes enfoques. Lo que tienen en común es que no son relacionales. Su primera ventaja es que, a diferencia de las bases de datos relacionales, manejan eficazmente datos no

estructurados como ficheros texto, correo electrónico, multimedia y los medios de comunicación sociales (Leavitt, 2010).

Algunas bases de datos NoSQL pueden funcionar de manera distribuida. Los usuarios pueden, de esta manera, gestionar una base de datos ejecutándola en varias computadoras económicas en vez de una con más potencia, pero más costosa (Leavitt, 2010).

Es más, los defensores dicen que las bases de datos NoSQL tienen un mejor rendimiento, que es particularmente importante para las aplicaciones con grandes cantidades de datos. Numerosas compañías y organizaciones han desarrollado bases de datos NoSQL. El enfoque más influyente es, principalmente, de las compañías campeones de la Web 2.0 con grandes y crecientes cantidades de datos como Amazon y Google. Ellos desarrollaron las bases de datos *NoSQL Dynamo* y *BigTable* respectivamente, que ha inspirado muchas de las aplicaciones NoSQL de hoy (Leavitt, 2010).

La Tabla 5 muestra una comparación entre algunas de las bases de datos NoSQL.

Características	**DynamoDB**	**Cassandra**	**HBase**	**MongoDB**	**CouchDB**	**BigTable**
Tipo almacenamiento	Llave-valor	Llave-valor	Llave-valor	Orientado a documento	Orientado a documento	Orientado a columna
Lanzamiento inicial	2012	2008	2010	2009	2005	2005
Consistencia	S/R	S/R	Si	Si	S/R	Si
Tolerancia al particionado	S/R	Si	Si	Si	Si	Si
Persistencia	Soporte	Soporte	Soporte	Soporte	S/R	Soporte
Alta disponibilidad	Soporte	Soporte	Soporte	Soporte	S/R	Soporte
Durabilidad	Soporte	Soporte	Soporte	Soporte	Soporte	Soporte
Escalabilidad	Alto	Alto	Alto	Alto	Alto	Alto
Desempeño	Alto	Alto	Alto	Alto	Alto	Alto
Esquema libre	Soporte	Soporte	Soporte	Soporte	Soporte	Soporte
Lenguaje de programación	Java	Java	Java	C++	Erlang	C, C++
Plataforma	Linux	Windows, Linux, OS X	Windows, Linux, OS X	Windows, Linux, OS X	Windows, Linux, OS X	Windows, Linux, OS X
Código abierto	Sin soporte	Soporte	Soporte	Soporte	Soporte	Sin soporte
Desarrollador	Amazon	ASF	ASF	10gen	ASF	Google

Tabla 5: Comparación de algunas bases de datos NoSQL (Hashem et al., 2014).

1.5.1.1 MongoDB

MongoDB es un sistema gestor de bases de datos no relacional, multiplataforma e inspirada en el tipo de bases de datos documental, su nombre proviene del término en inglés

"huMONGOus" (enorme). Está liberada bajo licencia de software libre, específicamente GNU AGPL 3.0. MongoDB usa el formato BSON (JSON Binario) para guardar la información, dando la libertad de manejar un esquema libre.

El desarrollo de MongoDB comenzó en el año 2007 por la empresa 10gen publicando una versión final en el 2009. Está escrito en C++, posee algunas características SQL y replicación maestro-esclavo.

Según (Chodorow, 2013), algunos términos básicos entorno a MongoDB son:

- Un documento es la unidad básica de información para MongoDB y es apenas equivalentes a una fila en un sistema de gestión de base de datos de relaciones.
- De modo semejante, una colección puede ser considerada como una tabla, pero con un esquema dinámico.
- Una sola instancia de MongoDB puede administrar múltiples bases de datos independientes, cada una de ellas puede tener sus propias colecciones.
- Cada documento tiene una llave especial denominada "_id" que es única dentro de una colección.
- MongoDB viene con un simple pero poderoso *shell* de JavaScript, lo cual es útil para la administración de instancias de MongoDB y la manipulación de datos.

1.5.1.2 Ventajas de las NoSQL.

Según (Leavitt, 2010) las bases de datos NoSQL generalmente procesan los datos más rápido que las bases de datos relacionales. Estas últimas normalmente son usadas para los negocios y a menudo para transacciones que requieren de gran precisión. Generalmente se someten los datos al conjunto de restricciones ACID[2] (atomicidad (*atomicity*), consistencia (*consistency*), aislamiento (*isolation*) y durabilidad (*durability*)).

Al tener que realizar estas restricciones para cada dato hace de las bases de datos relacionales más lentas. Los desarrolladores usualmente no tienen en sus bases de datos NoSQL soporte ACID, para incrementar el rendimiento, pero esto puede causar problemas cuando se usen aplicaciones que requieran gran precisión (Leavitt, 2010).

[2] En bases de datos se denomina ACID a un conjunto de características necesarias para que una serie de instrucciones puedan ser consideradas como una transacción.

1.5.1.2 Preocupaciones e incertidumbres.

Las bases de datos NoSQL enfrentan numerosos desafíos o desventajas como los descritos en la Tabla 6.

Aspecto	**Descripción**
Gastos y complejidad	Como las bases de datos NoSQL no trabajan con SQL, se requiere una programación manual de la consulta, que puede ser rápido para tareas simples pero consume mucho tiempo para otras. Además, la programación de las consultas complejas puede ser muy difíciles
Seguridad	Las bases de datos relacionales, nativamente, soportan ACID, mientras las bases de datos NoSQL no lo hacen. Las bases de datos NoSQL no ofrecen así el grado de fiabilidad que ACID proporciona. Si los usuarios quieren aplicar las restricciones ACID a un juego de datos en una base de datos NoSQL, deben realizar una programación adicional.
Consistencia	Como las bases de datos NoSQL no soporta las transacciones ACID nativamente, ello también podría comprometer la consistencia, a menos que se proporcione soporte manual. No soportar la consistencia habilita mejor rendimiento y escalabilidad, pero es, con toda seguridad, un problema para algunos tipos de aplicaciones y transacciones como las involucradas con las bancarias.
No familiaridad con la tecnología	Muchas organizaciones no están familiarizadas con bases de datos NoSQL y no se sienten con el conocimiento suficiente para escoger una o incluso para determinar que enfoque podría ser mejor para sus propósitos.
Soporte limitado	A diferencia de las bases de datos relacionales, muchas aplicaciones NoSQL de código abierto no poseen herramientas de soporte o gestión para el cliente.

Tabla 6: Desventajas de las NoSQL (Leavitt, 2010).

1.5.2 Hadoop

Hadoop fue creado por Doug Cutting, que lo nombró así por un elefante de juguete. Hadoop es un proyecto Apache de alto nivel que está siendo usado por una comunidad global de contribuidores. Fue desarrollado originalmente para apoyar la distribución del proyecto de motor de búsqueda Nutch.

Está inspirado en el proyecto de Google File System (GFS) y en el modelo de programación MapReduce, el cual consiste en las tareas *mapper* y *reducer* para manipular los datos, distribuidos en nodos de un clúster y logrando un alto paralelismo en el procesamiento.

Hadoop llena un vacío en el mercado, almacenando y proporcionando capacidades computacionales eficazmente sobre cantidades sustanciales de datos. Es un sistema

distribuido compuesto de un sistema del archivo distribuido y ofrece una vía para paralelizar y ejecutar programas en un clúster de máquinas, ver Figura 6 (Holmes, 2012).

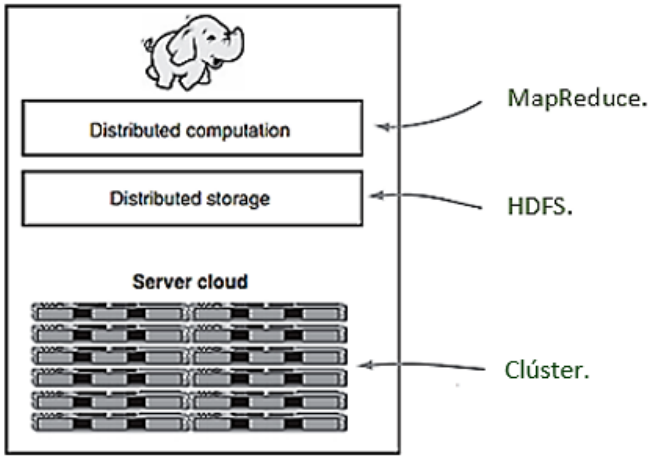

Figura 6: Ambiente de Hadoop (Holmes, 2012).

Hadoop, como se muestra en la Figura 7, es una arquitectura distribuida maestro-esclavos (*master-slaves*) que consiste en el sistema de archivos distribuidos *Hadoop Distributed File System* (HDFS, por sus siglas en inglés) para la capa de almacenamiento, MapReduce para las capacidades computacionales, es decir, en la capa de procesamiento y Hadoop Common para complementar en núcleo de Hadoop. Los rasgos intrínsecos de Hadoop son el particionado de los datos y la computación paralela en conjuntos grandes de datos (Holmes, 2012).

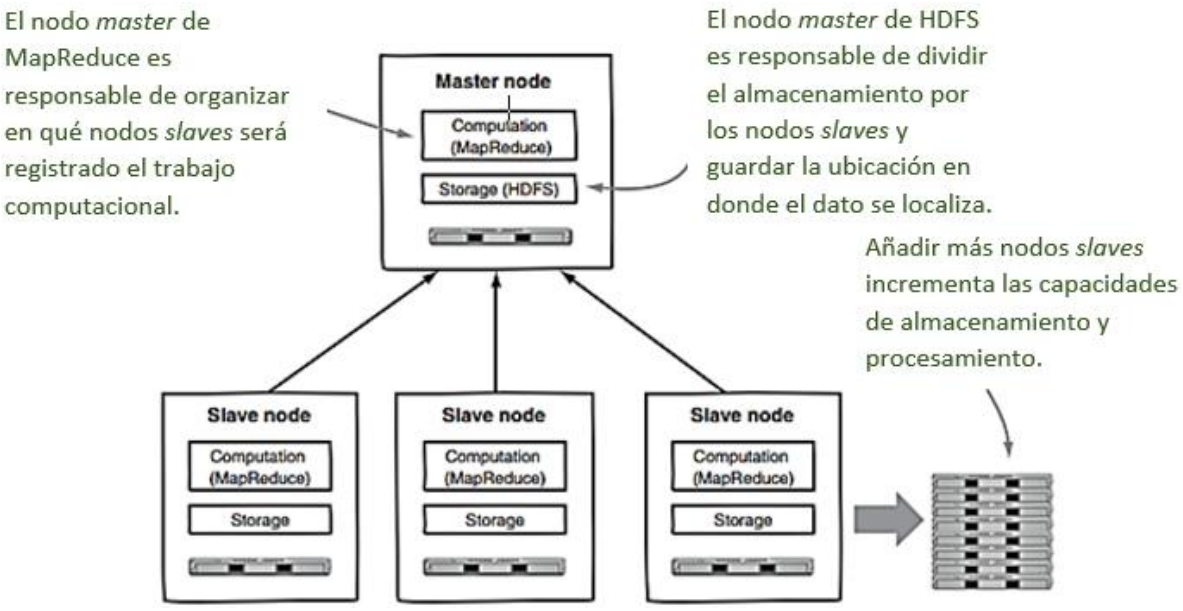

Figura 7: Arquitectura Hadoop de alto nivel (Holmes, 2012).

1.5.2.1 HDFS

HDFS es el componente del almacenamiento de Hadoop. Es un sistema del archivo distribuido basado en GFS. HDFS está optimizado para los grandes procesamientos y trabaja mejor cuando lee y escribe grandes archivos (gigabytes y más grande). Para apoyar este procesamiento, HDFS contiene bloques de gran tamaño (para un sistema del archivo) y optimizaciones locales de datos para reducir la entrada/salida de la red (Holmes, 2012) .

Gracias, en parte, a la replicación de los datos y la tolerancia al fallo se logra la escalabilidad y disponibilidad, los cuales constituyen también rasgos importantes de HDFS. Además, replica los archivos un número de veces, configurado por el usuario, por defecto tres. Es tolerante al fallo del software y hardware, y automáticamente replica los bloques de datos de los nodos que han fallado (Holmes, 2012).

La Figura 8 muestras una representación lógica de los componentes de HDFS: *NameNode* y *DataNode*. También se observa una aplicación que está usando la biblioteca del sistema de archivo de Hadoop para acceder a HDFS (Holmes, 2012).

En el libro (Venner, 2009) se explica que el *NameNode* maneja el almacenamiento de los metadatos del sistema del archivo. Esto incluye los caminos del archivo, los bloques que constituyen los archivos y el *DataNode* que sostiene los bloques. El *NameNode* escribe un diario de entradas para editar los ficheros *logs* cuando se realizan cambios. El *NameNode* mantiene el conjunto de datos entero en memoria para permitir un tiempo de la respuesta más rápido para las demandas que no involucran una modificación.

Los *DataNodes* almacenan, recuperan y eliminan los bloques de los datos. Según (Venner, 2009), el tamaño del bloque básico es de 64 MB. A menos que un archivo (como un los archivos con extensión *.zip*, *.tar*, *.tgz*, *.tar.gz* o *.har*) sea usado, no se compartirán los bloques entre los múltiples archivos.

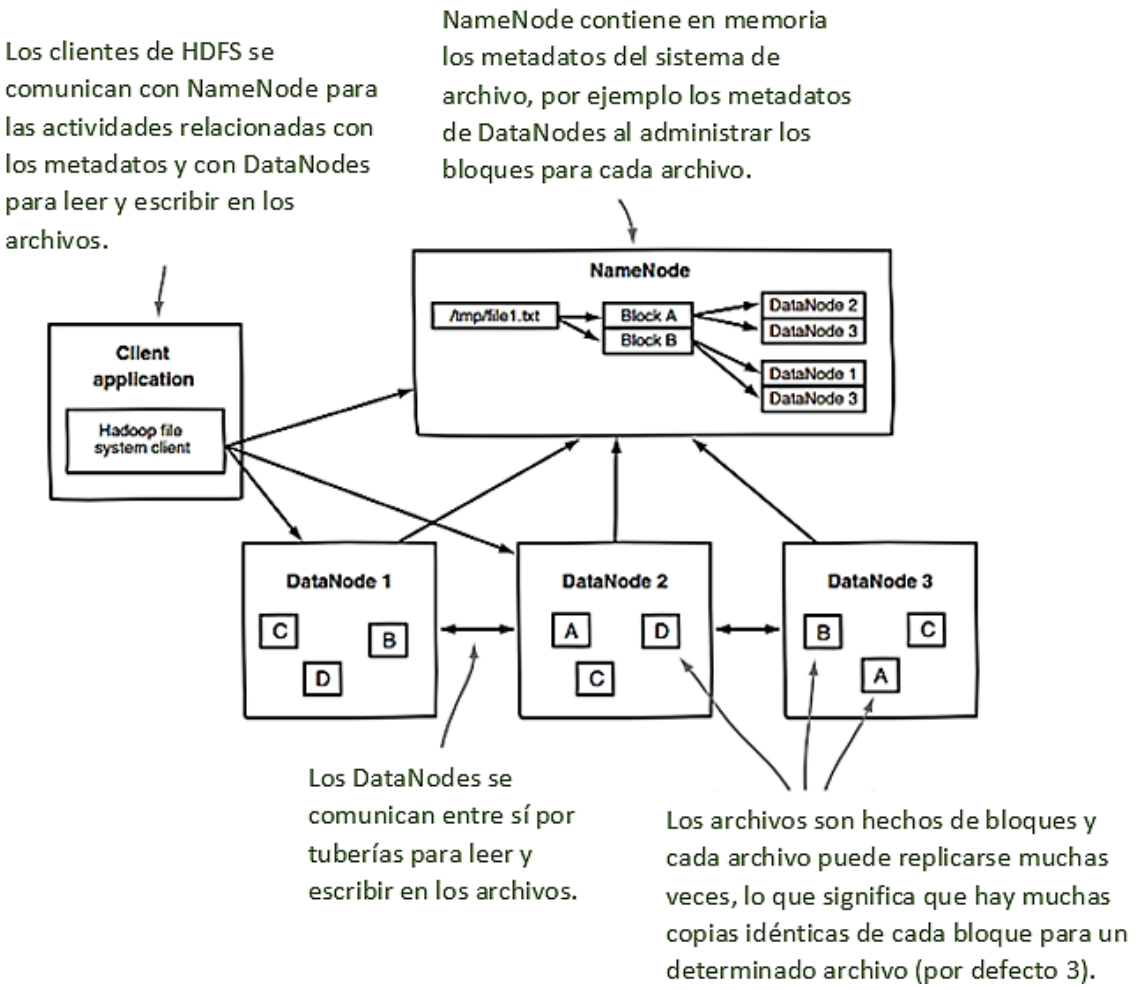

Figura 8: Arquitectura HDFS mostrando la comunicación del cliente con el maestro (NameNode) y los esclavos (DataNodes) (Holmes, 2012).

1.5.2.2 Hadoop MapReduce

MapReduce es un modelo de programación para procesar y generar *big data*. Está inspirado por las funciones *map* y *reduce* presentes en Lisp y otros lenguajes funcionales. Los usuarios especifican una función *map* que procesa un par llave/valor para generar un conjunto intermedio de llaves/valores y la función *reduce* fusiona todos los valores intermedios asociados con la misma llave intermedia. Muchas tareas del mundo real pueden ser expresadas mediante este en este modelo (Dean and Ghemawat, 2008).

Los programas escritos en este estilo funcional son automáticamente paralelizados y ejecutados en un clúster de máquinas. El sistema se encarga de dividir los datos de entrada, planificando la ejecución del programa en un conjunto de máquinas, ocupándose de las fallas y manejando la comunicación requerida entre las máquinas. Esto permite a los

programadores sin experiencia con este tipo de sistema utilizar fácilmente los recursos de un gran sistema distribuido (Dean and Ghemawat, 2008).

El papel de los programadores es, básicamente, definir las funciones *map* y *reduce*, donde la función *map* devuelve como salida un conjunto de tuplas llave/valor, las cuales son procesadas por la función *reduce* para producir la salida final. En la Figura 9 se muestra un fragmento escrito en pseudocódigo de la función *map* considerando su entrada y salida, para un caso genérico (Holmes, 2012).

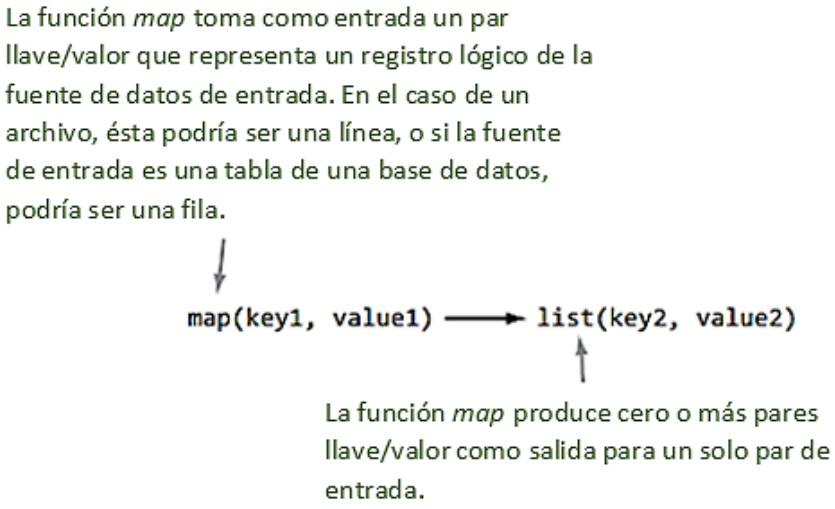

Figura 9: Vista lógica de la función map (Holmes, 2012).

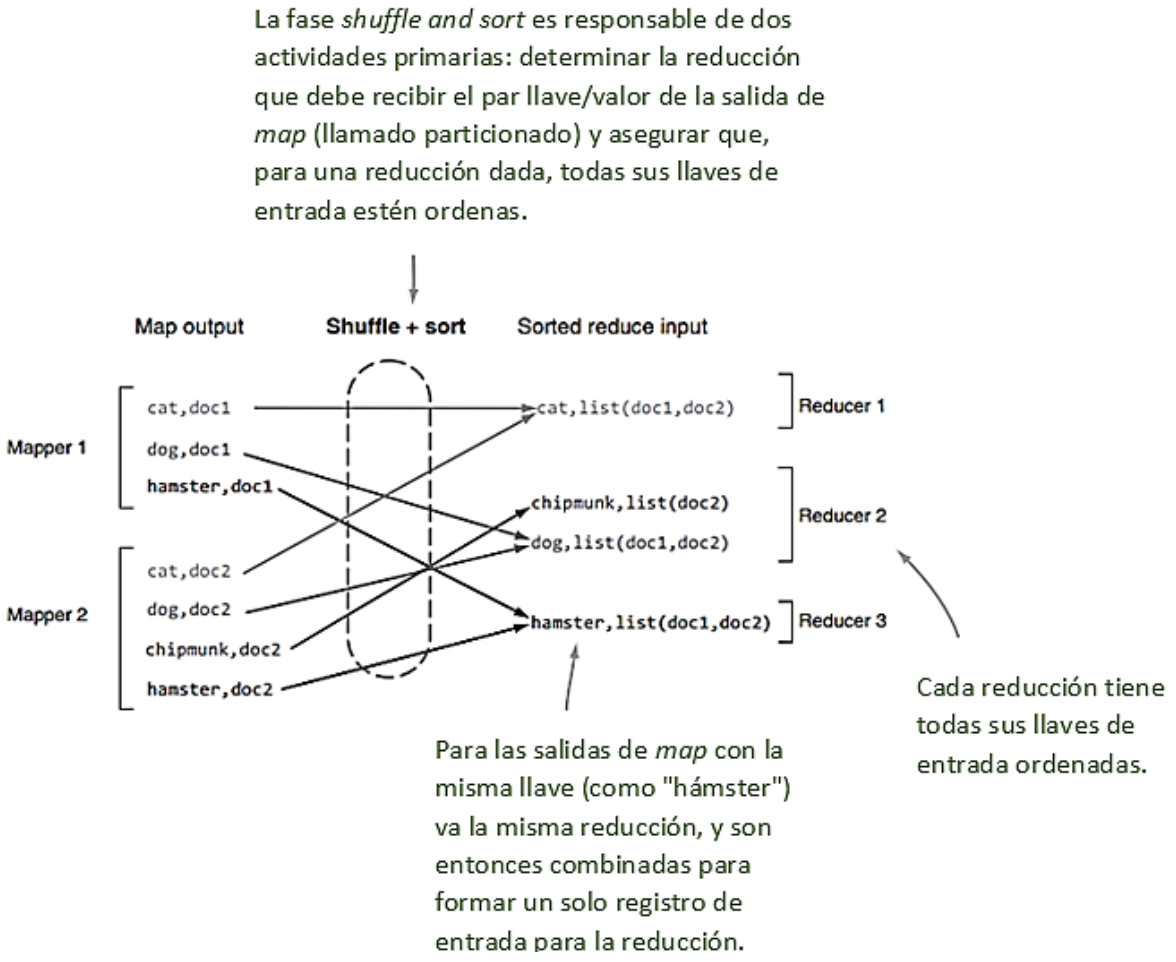

Figura 10: Shuffle and sort (mezclar y ordenar) de MapReduce (Holmes, 2012).

La mayor potencialidad de MapReduce ocurre entre la salida de *map* y la entrada de *reduce*, en la fase de *shuffle and sort* (mezclar y ordenar), como se observa en la Figura 10. La Figura 11 muestra un fragmento en pseudocódigo de la función *reduce*. La arquitectura MapReduce de Hadoop es similar al modelo del *master-slave* en HDFS. Los componentes principales de MapReduce se ilustran en su arquitectura lógica, como se muestra en la Figura 12.

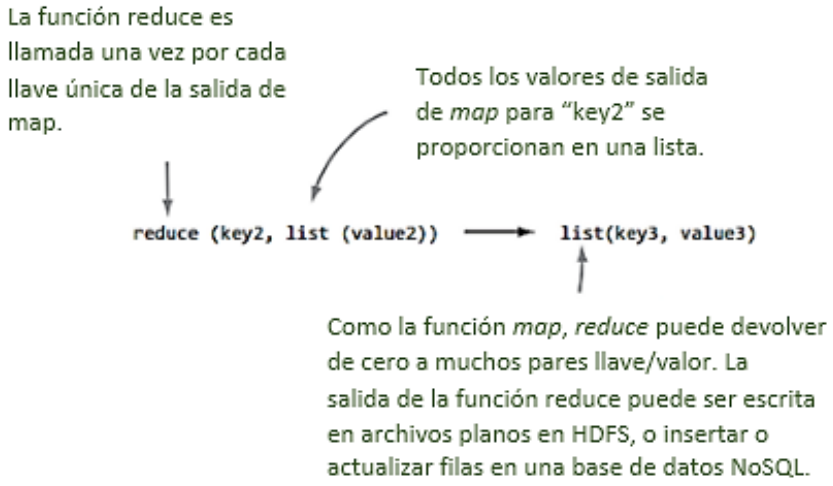

Figura 11: Vista lógica de la función reduce. (Holmes, 2012).

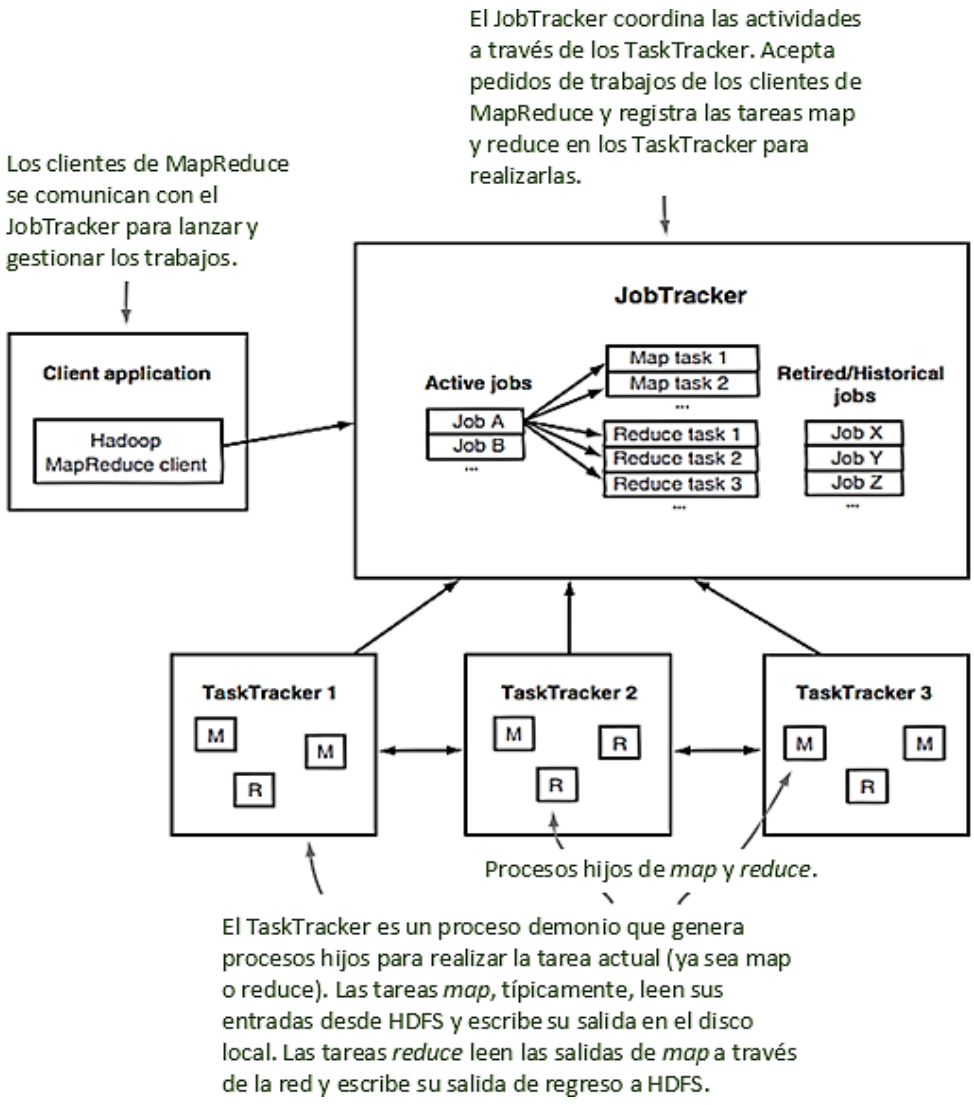

Figura 12: Arquitectura lógica de MapReduce (Holmes, 2012).

Por ejemplo, considerando el problema de contar el número de ocurrencias de cada palabra en una colección grande de documentos. Según (Dean and Ghemawat, 2008) el usuario puede escribir la función *map* y *reduce* de manera similar al pseudocódigo de la Figura 13:

```
map(String name, String document):
 // llave: nombre del documento
 // valor: contenido del documento
 for each word w in document:
    EmitIntermediate(w, 1);

reduce(String word, Iterator partialCounts):
  // word: una palabra
 // partialCounts: una lista parcial de la cuenta agregada
 int result = 0;
 for each v in partialCounts:
    result += ParseInt(v);
 Emit(result);
```

Figura 13: Pseudocódigo de WordCount.

La función del *map* emite cada palabra más una cuenta asociada de ocurrencias (sólo 1 en este ejemplo sencillo). La función *reduce* suma todas las cuentas emitidas para una palabra particular. Además, el usuario escribe el código para rellenar la especificación de un objeto MapReduce con los nombres de los archivos entrada y salida, y parámetros opcionales para la puesta a punto. Entonces el usuario invoca la función MapReduce pasándole la especificación del objeto (Dean and Ghemawat, 2008).

Cuando el programa llama una función MapReduce, ocurre la siguiente secuencia de acciones (Dean and Ghemawat, 2008) (ver Figura 14):

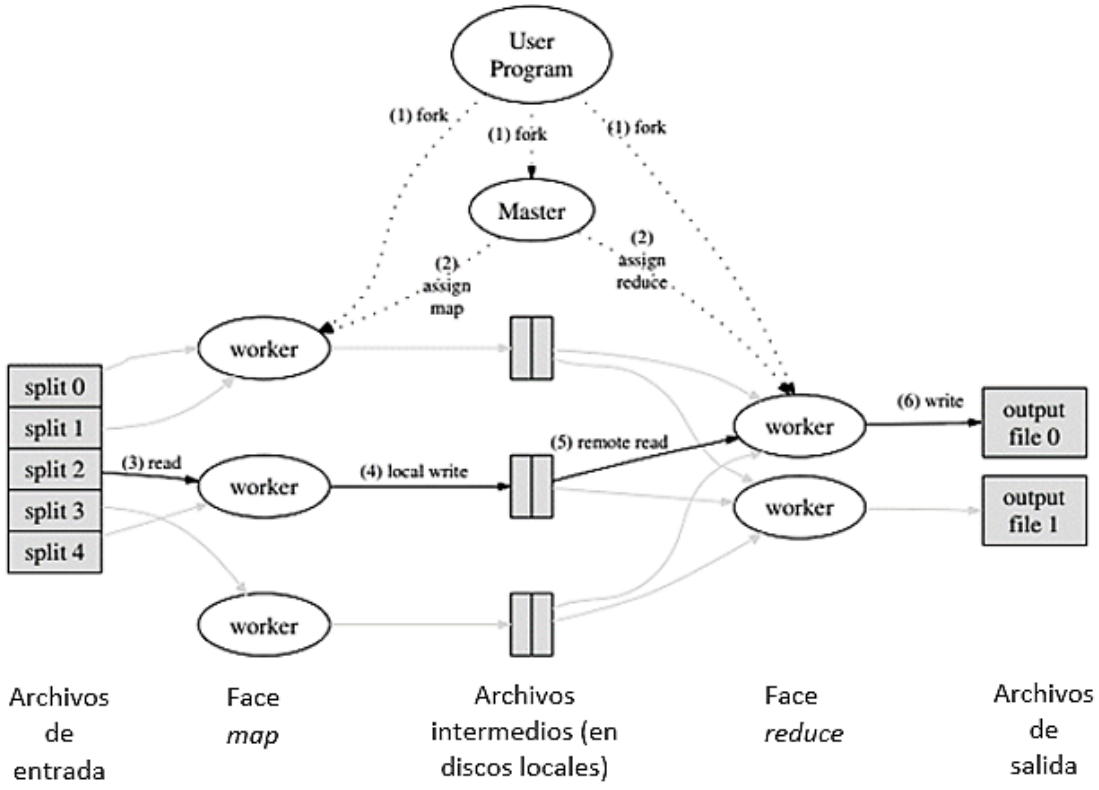

Figura 14: Perspectiva general de la ejecución (Dean and Ghemawat, 2008).

1. La biblioteca de *MapReduce*, en el programa del usuario, primero divide los archivos de entrada en M pedazos, típicamente de 16 a 64 megabytes cada uno (esta acción es controlable por el usuario a través de un parámetro opcional). Luego pone en marcha muchas copias del programa en un clúster de máquinas.
2. Una de las copias del programa es especial, el *master*. Las restantes son *workers* que tienen trabajo asignado por el *master*. Hay M tareas *map* y *R reduce* para asignar. El *master* escoge los *workers* que están ociosos y le asigna a cada uno una tarea de *map* o *reduce*.
3. Al *worker* que se le asignó una tarea *map* lee el contenido de la entrada correspondiente. Analiza gramaticalmente los pares llave/valor fuera de datos de entrada y pasa cada par a la función *map* definida por el usuario. Los pares llave/valor intermedios producidos por la función *map* se almacenan en memoria.
4. Periódicamente, los pares almacenados se escriben en el disco local, particionado en R regiones por la función de particionado. La ubicación de estos pares almacenado

en el disco local se le pasa al *master*, que es el encargado de enviar estas ubicaciones a los *workers* que hacen las tareas *reduce*.

5. Cuando un worker tiene una tarea *reduce* y es notificado por el *master* sobre estas ubicaciones, usa llamadas a procedimientos remotos para leer los datos almacenados en los discos locales de los *workers* de mapeo. Cuando un *worker* de reducción ha leído todos los datos intermedios, los ordena por las llaves intermedias para que se agrupen todas las ocurrencias de la misma llave. El ordenamiento se necesita porque hay muchas llaves diferentes derivadas de *map* para una misma tarea de reducción. Si la cantidad de datos intermedios es demasiado grande para caber en la memoria, se usa un ordenamiento externo.
6. Los *worker*s de reducción iteran sobre los datos intermedios ordenados y por cada llave intermedia encontrada, pasa la llave y el conjunto de valores intermedios correspondiente a la función *reduce* del usuario. La salida de la función *reduce* se anexa a un archivo de salida final para esta partición
7. Cuando todas las tareas *map* y *reduce* se han completado, el *master* despierta el programa del usuario. En este momento, la llamada a MapReduce en el programa del usuario regresa al código del usuario.

1.5.2.3 Hadoop Common

Son un conjunto de bibliotecas que soportan varios proyectos de Hadoop. Se puede decir que es la solución tecnológica sobre el procesamiento de *big data* que más se destaca. Cabe especificar que Hadoop no es un programa en sí, es decir, no podemos descargar un programa denominado Hadoop directamente, ya que es un ecosistema de productos bajo la licencia Apache Software Foundation. De esta forma hay dos productos principales que conforman el núcleo de cualquier aplicación Hadoop. Estos son el HDFS y MapReduce. Pero además de estos productos básicos, existe una multitud de productos o iniciativas de código abierto que modifican o complementan el núcleo de Hadoop, algunos de los cuales se observan en la Tabla 7.

Referencia	**Biblioteca**	**Descripción**
(Thusoo et al., 2009)	Hive	Hive ofrece una estructura de almacenes de datos en HDFS.
(George, 2011)	Hbase	Base de datos distribuida y escalable que apoya el almacenamiento de los datos estructurados para tablas grandes.
(Owen et al., 2011)	Mahout	Mahout es una biblioteca de aprendizaje automatizado y

		minería de datos. Comparado con otros algoritmos preexistentes, la biblioteca de Mahout pertenece al subconjunto que puede ejecutarse de manera distribuida y es ejecutable por MapReduce.
(Olston et al., 2008)	Pig	El marco de trabajo Pig involucra un lenguaje script de alto nivel (Pig Latin) y ofrece una plataforma en tiempo de ejecución que les permite a los usuarios ejecutar MapReduce en Hadoop.
(Zaharia et al., 2010)	Spark	Un motor rápido y de cálculo general para datos de Hadoop.
(Rabkin and Katz, 2010)	Chukwa	Marco de trabajo de análisis incorporado con MapReduce y HDFS. El flujo de datos de Chukwa tiene en cuenta la colección de los datos de los sistemas distribuidos, datos procesados y datos almacenados en Hadoop. Como un módulo independiente, Chukwa es incluido en la distribución de Apache Hadoop.
(Vavilapalli et al., 2013)	YARN	Siguiente generación de Hadoop. Separa las funciones de gestión de recursos del modelo de programación.
(Hoffman, 2013)	Avro	Las tareas realizadas por Avro incluyen serialización de los datos, llamadas a procedimientos remotos y datos que pasan de un programa o lenguaje a otro. Este software es conveniente para aplicaciones de lenguajes script, como Pig, debido a sus cualidades.
(Dirk deRoos and Melnyk, 2014)	Sqoop	Una herramienta eficiente para la transferencia de datos de una base de datos relacional al HDFS.
(Dirk deRoos and Melnyk, 2014)	Flume	Un servicio de flujo de datos para el movimiento de grandes volúmenes de datos log hacia Hadoop.
(Dirk deRoos and Melnyk, 2014)	Ambari	Un conjunto integrado de herramientas de administración para instalar, supervisar y mantener un clúster de Hadoop. También se incluyen herramientas para agregar o quitar los nodos esclavo.
(Dirk deRoos and Melnyk, 2014)	HCatalog	Servicio que provee una vista relacional de los datos almacenados en Hadoop, incluyendo el enfoque estándar para datos tabulares.
(Dirk deRoos and Melnyk, 2014)	Hue	Interfaz de administración de Hadoop con para buscar archivos, resultados de consultas de Pig y Hive y desarrollar flujos de datos en Oozie.
(Dirk deRoos and Melnyk, 2014)	Oozie	Herramienta de gestión de flujos de datos que puede ocuparse de la planificación y concatenación de las aplicaciones de Hadoop.
(Dirk deRoos and Melnyk, 2014)	ZooKeeper	Una interfaz simple para la coordinación centralizada de servicios (como nombrar, configurar y sincronizar) usado para aplicaciones distribuidas.
(Radia and Srinivas, 2014)	Tez	Permite ejecutar programas Hive y Pig más natural, como un solo trabajo en lugar de múltiple fases de MapReduce, produciendo mejoras en el rendimiento.

Tabla 7: Bibliotecas de Hadoop.

Un punto que es necesario tener claro es que Hadoop es un conjunto de programas o plataformas, que son capaces de colaborar entre sí y crear asociaciones, debido a sus diferentes funcionalidades.

La Figura 15 muestra el variado ecosistema de proyectos de Hadoop y cómo se relacionan unos con otros.

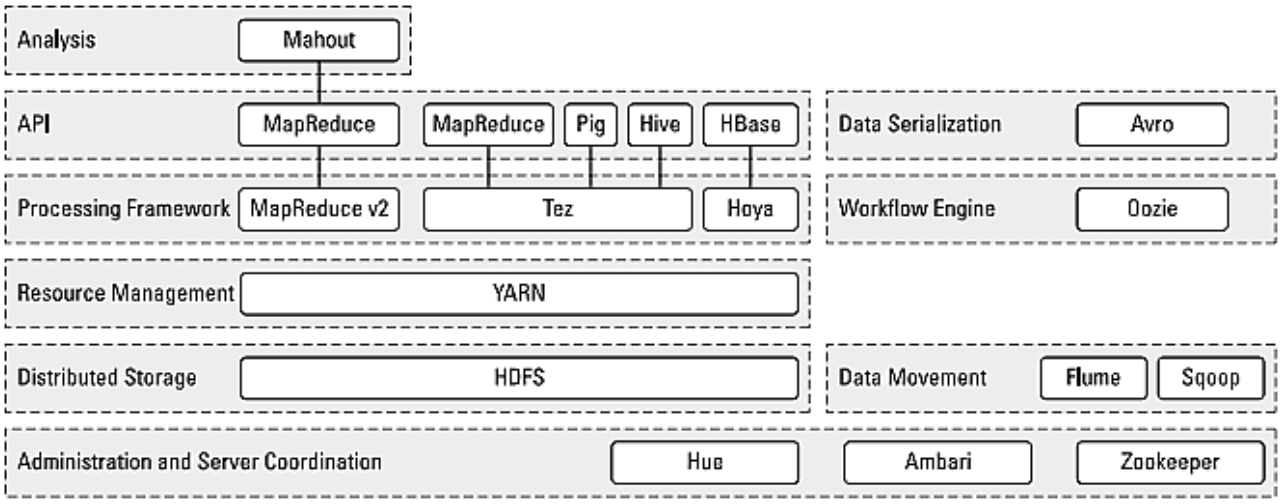

Figura 15: Componentes del ecosistema de Hadoop (Dirk deRoos and Melnyk, 2014).

1.5.2.4 Evolución de la versión 1 de Hadoop a la 2

La versión 2 de Hadoop contiene cambios fundamentales en la arquitectura que extienden significativamente la plataforma, desarrollando la capa de procesamiento más allá de MapReduce e introduciendo nuevos paradigmas en la aplicación. Análogamente, el subsistema de almacenamiento se ha generalizado para apoyar otros marcos de trabajo además de HDFS. La nueva versión mejora significativamente la escalabilidad y rendimiento en las capas de almacenamiento y procesamiento, esta última puede procesar hasta 100 mil tareas de manera concurrente.

Evolución de la arquitectura en la capa de almacenamiento:

Inicialmente en las primeras versiones de Hadoop, los recursos de almacenamiento de un clúster sólo estaban disponibles en HDFS. Con la llegada de la versión dos, una nueva arquitectura de almacenamiento es generalizada para que no sólo pueda ser usada por HDFS sino también por otros servicios de almacenamiento. El primer uso de este rasgo es que permite múltiples instancias del espacio de nombres de HDFS para compartir el almacenamiento subyacente (Radia and Srinivas, 2014).

Otro cambio fundamental de almacenamiento es que se comienza a trabajar en el almacenamiento heterogéneo. La versión 1 trata todos los dispositivos de almacenamiento sobre un *DataNode*. El almacenamiento heterogéneo es parte del lanzamiento de la versión 2 en el 2014, donde el sistema distingue entre los tipos de almacenamiento y también hace disponible la información almacenada a otros marcos de trabajo y aplicaciones para que ellos puedan sacar ventaja de las propiedades del almacenamiento (Radia and Srinivas, 2014). Algunas de estas nuevas características se aprecian en la Tabla 8.

Característica	**Descripción**
Alta disponibilidad del *NameNode*	En Hadoop 1 la capa de almacenamiento HDFS es totalmente distribuida y tolerante al fallo, además los metadatos del sistema de archivo se guardan en un solo servidor maestro llamado *NameNode*. Cuando el *NameNode* se apaga por un mantenimiento planeado o por un fallo en el sistema, el clúster no se encuentra disponible hasta que se reinicie el *NameNode*. Hadoop 2 agrega apoyo con un *NameNode* en estado de espera En caso de fracaso del *NameNode* activo, la recuperación automática se activa y el *NameNode* de reserva se pone en funcionamiento.
Controlador de recuperación de fallos	Un nuevo proceso llamado *ZKFC* (*ZooKeeper-based Failover Controller*, controlador de recuperación de fallo basado en ZooKeeper) gestiona la recuperación de fallos de los *NameNodes*. Este proceso ejecuta en cada uno de los *NameNodes* y mantiene una sesión con el *ZooKeeper*. Al usar *ZooKeeper* para la coordinación, uno de los ZKFC se vuelve el líder y elige el *NameNode* local como activo. El *ZKFC* realiza periódicamente un chequeo al *NameNode*. Cuando el *NameNode* activo falla, *ZooKeeper* descubre la pérdida y elimina el *ZKFC* del nodo que falló como líder. Esto produce la recuperación de fallo automático; el *ZKFC* que se ejecuta en estado de espera se vuelve líder y se activa el *NameNode* de reserva.
Soporte NFS (*Network File System*, sistema de archivos de red)	Normalmente se accede a HDFS a través de la biblioteca del cliente HDFS o por HTTP. La falta de integración con el sistema de archivos del cliente hace difícil para los usuarios e imposible para algunas aplicaciones, acceder a HDFS. Hadoop 2 agrega NFS versión 3 para hacer esta integración de manera fácil.

Instantáneas de HDFS	Hadoop 2 agrega soporte para tomar instantáneas al sistema de archivos. Una instantánea es una imagen de un punto en el tiempo del sistema del archivo entero, la cual tiene usos como: • Protección contra los errores del usuario: • Copia de seguridad: • Recuperación de un desastre: Las características de las instantáneas ofrecen ser tomadas en modo de sólo lectura y sólo se lleva a cabo en el *NameNode*, y ninguna copia de datos es hecha cuando se toma una.
Mejoras en la entrada y salida de los datos.	Las mejoras a HDFS aceleran y se agregan eficiencia. Los clientes pueden leer directamente del sistema de archivo local en lugar de un *socket* del *DataNode*. Dichas mejoras han hecho que la entrada o salida se ejecuten de 2.5 a 5 veces más rápidas que en lanzamientos anteriores.

Tabla 8: Evolución en la capa de almacenamiento de Hadoop (Radia and Srinivas 2014).

Evolución de la arquitectura en la capa de procesamiento:

Previamente, los recursos computacionales en Hadoop sólo estaban disponibles para los programas y aplicaciones MapReduce. El componente YARN (*Yet Another Resource Negotiator*, otro negociador de recursos) generaliza la capa de procesamiento para no ejecutar solamente el modelo MapReduce sino también marcos de trabajos de otras aplicaciones. Como resultado, el YARN permite ejecutar consultas analíticas significativamente más eficientes y ha permitido una nueva variedad de aplicaciones. La nueva arquitectura es más descentralizada y permite al clúster Hadoop ser escalado con más microprocesadores y servidores de manera significativa, ver Tabla 9.

Característica	**Descripción**
Gestión de recursos descentralizados	Hadoop 1 tiene un solo servidor maestro llamado *JobTracker* para gestionar los recursos computacionales y los trabajos que usan los recursos. YARN divide esta función para que el *Resource Manager* (Gerente de Recursos) se enfoque en gestionar los recursos del clúster y el *Application Master* (Maestro de Aplicación), maneja cada aplicación que se ejecuta (como un trabajo de MapReduce). El *Application Master* solicita los recursos del *Resource Manager* basado en las necesidades y características de la aplicación que se ejecuta.
Soporte de primera clase para diferentes tipos de aplicaciones.	Como el *Application Master* está separado del *Resource Manager*, puede personalizarse para cada tipo de aplicación. Hadoop 2 tiene un *Application Master* especializado para MapReduce y otro marco de trabajo más generalizado llamada *Tez* que permite grafos acíclicos dirigidos *(DAGs*, por sus siglas en inglés) de ejecución. Nuevas variedades de aplicaciones para procesamiento por flujo, como *Samza* y *Storm* en YARN, también se

	ejecutan como aplicaciones de primera clase. Esto permite una consolidación del clúster y los recursos computacionales para ejecutar aplicaciones heterogéneas, resultando en una menor fragmentación de los recursos y una utilización más eficaz de los mismos.

Tabla 9: Evolución en la capa de almacenamiento de Hadoop (Radia and Srinivas 2014).

Como resumen se expone en la Figura 16, los cambios fundamentales en la arquitectura de Hadoop.

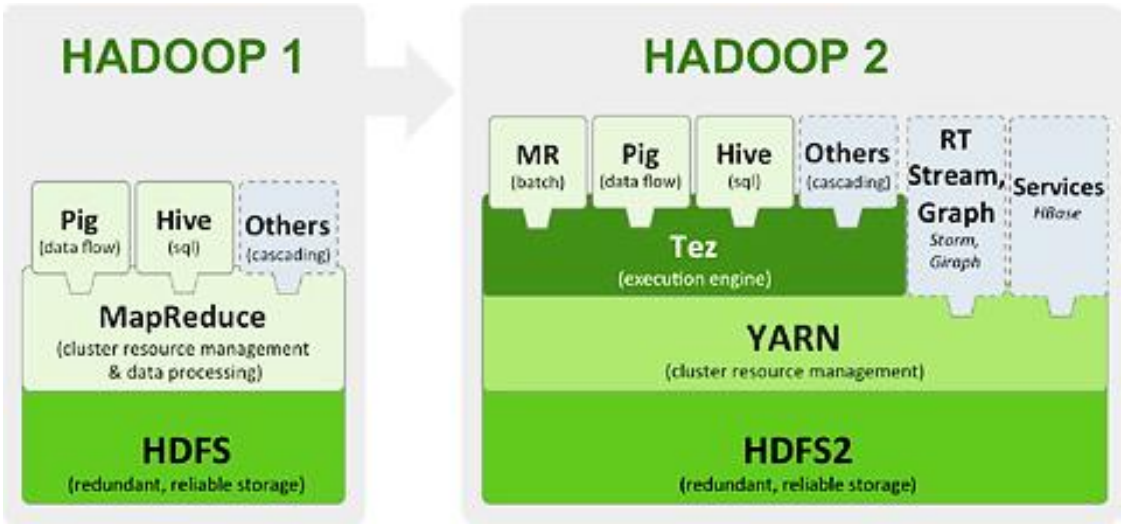

Figura 16: Comparación entre las arquitecturas de Hadoop versión 1 y 2.

1.5.3 Fases de big data y su relación con bibliotecas Hadoop

De acuerdo a las fases de *big data* previamente expuestas, las bibliotecas creadas entorno Hadoop que corresponden con cada una son:

- Adquisición de datos y grabación: En función del tipo y el origen de los datos se puede encontrar varias bibliotecas incluyen en esta fase. Si están presentes datos no estructurados, como pueden ser ficheros *logs*, los marcos de trabajo más utilizados son Flume y Chukwa. Sin embargo si se está tratando con datos provenientes de una base de datos relacional, Sqoop es la herramienta más utilizada.
- Extracción y preprocesamiento de la información: En esta fase quizá coincidirían las bibliotecas de las que se ha hablado en la fase anterior. Hay que tener en cuenta que aparte de grabar/almacenar los datos estas bibliotecas tienen opciones para hacer un filtrado previo de los mismos dando además de la estructura conveniente para que sirvan de entrada a Hadoop.

- Representación, agregación e integración de datos: En esta fase se incluyen varias bibliotecas o sistemas que no tienen mucho que ver en cuanto a sus funciones, pero pueden enmarcar en esta fase:
    a. HDFS: Previamente al tratamiento de la información por parte de Hadoop se necesita que los datos preprocesados sean almacenados en un sistema distribuido de ficheros. Este es el rol que cumple HDFS como componente dentro del núcleo de Hadoop.
    b. Avro: Como se dijo anteriormente, se trata de un sistema de serialización de datos que permite codificar los datos que va a manejar Hadoop. Permite definir los datos a serializar, además de definir interfaces a la hora de analizar semánticamente la información.
- Procesamiento de peticiones, modelado de datos y análisis: Dentro de esta fase se incluyen dos tipos de *softwares* distintas, las bases de datos y marcos de trabajos para ejecutar consultas sobre las bases de datos.
    a. Bases de datos: Dentro de estas bases de datos se puede HBase.
    b. Marcos de trabajos de consultas: Los dos más conocidos en este apartado son Hive y Pig. Aunque sirven son similares en cuanto a su uso, tienen dos enfoques distintos. Hive permite crear tablas, insertar datos y realizar consultas con un lenguaje similar a como se podría realizar consultas SQL. La diferencia fundamental con SQL es que utiliza MapReduce para ejecutar todas esas consultas. Por otro lado Pig permite manejar datos mediante un lenguaje textual propio de sentencias llamado Pig Latin. Este lenguaje permite paralelizar la ejecución de sentencias para crear, consultar o listar datos de manera sencilla mediante comandos predefinidos.
- Interpretación de datos: Dentro de esta fase quizá la herramienta más conocida sea Mahout. Mahout es una biblioteca de minería de datos que permite realizar clusterización, algoritmos de regresión o implementar modelos estadísticos sobre los datos de salida ya procesados.

1.6 Conclusiones parciales

En este capítulo se analizó la arquitectura, principales características y funcionalidades de las herramientas que trabajan con *big data*. Además, se muestran entre otros aspectos un

acercamiento al término *big data* y a las herramientas que se relacionan con el mismo: definición del término, características fundamentales y tendencias actuales de las tecnologías que utiliza.

CAPÍTULO 2: CONFIGURACIÓN DE HADOOP Y MONGODB PARA EL CASO DE ESTUDIO SISTEMA DE GESTIÓN BIBLIOGRÁFICA ABCD.

En este capítulo se presentarán las características y especificaciones de la configuración a realizar en el marco de trabajo Hadoop, los casos de usos más frecuentes en la integración de Hadoop y MongoDB y una descripción del caso de estudio.

2.1 Instalación de Hadoop 2.6.0

Antes de instalar Hadoop se deben tomar tres decisiones:

- Escoger la distribución adecuada.
- Determinar las computadoras que formarán parte del clúster de Hadoop.
- Escoger el modo de configuración del clúster.

2.1.1 Escoger la distribución adecuada

Las distribuciones comerciales de Hadoop ofrecen varias combinaciones de componentes de código abierto de la Fundación de Software Apache. El objetivo de integrar varios componentes en un solo producto es quitarle al usuario el esfuerzo de ensamblar su propio juego de elementos integrados. Además del *software* de código abierto, los vendedores típicamente ofrecen *softwares* propietarios, soporte, consultas de servicios y entrenamiento (Dirk deRoos and Melnyk, 2014).

En la Tabla 10 se pueden apreciar las características básicas de tres distribuciones, pero el usuario es el que tiene que escoger. Dicha decisión se debe basar en un conjunto de criterios creados para tomar la mejor decisión posible.

Compañía	**Características**	**Clientes y contribuidores**
MapR	MapR presenta las características de MapRFS que es un sustituto de la norma HDFS. A diferencia de HDFS, este sistema tiene como objetivo mantener los clústeres que se componen de hasta 10 mil nodos sin un solo punto de fallo, que está garantizado por el *NameNode* distribuido. MapR permite almacenar de 1 a 10 exabytes de datos y proporciona soporte para NFS y la semántica de lectura y escritura aleatorias. Los desarrolladores de MapR declaran que la eliminación de las capas de	• Cisco Systems ha anunciado soporte para el software de MapR en la plataforma UCS. • comScore.

	abstracción de Hadoop puede ayudar a aumentar dos veces el rendimiento. Hay tres ediciones de la distribución MapR: M3, que es totalmente gratuito, M5 y M7, estos dos últimos son versiones empresariales pagadas. Aunque M3 ofrece escalabilidad ilimitada y NFS, no garantiza una alta disponibilidad e instantáneas, que están disponibles en M5 o las instantáneas de recuperación de M7. Este último es una plataforma de nivel empresarial para despliegues NoSQL y de Hadoop.	
Cloudera	La solución de Cloudera tiene las más poderosas herramientas de desarrollo y administración de Hadoop, diseñada para la gestión de un clúster de tamaño ilimitado. También es de código abierto y la empresa es un activo colaborador de Apache Hadoop. Además, la distribución Cloudera tiene sus propios componentes nativos, como el Impala, un motor de búsqueda basado en procesamiento paralelo masivo y Cloudera Search basado en Apache Solr.	• eBay • CBS Interactive • Qualcomm • Expedia
Hortonworks	Al ser 100% código abierto, Hortonworks está firmemente comprometido con Apache Hadoop y es uno de los principales contribuyentes a su solución. Stinger es una de las iniciativas de la empresa que trae un alto rendimiento, escalabilidad y compatibilidad de SQL para el clúster Hadoop. YARN y Apache Tez ayudan a Stinger a acelerar el procesamiento de Hive y Pig hasta 100 veces. Como resultado de la alianza de Hortonworks con Microsoft, HDP es la única distribución Hadoop disponible como un componente natural de Windows Server.	• Western Digital • eBay • Samsung Electronics

Tabla 10: Las tres distribuciones más prominentes (Starostenkov and Grigorchuk, 2013).

No todas las distribuciones de Hadoop tienen los mismos componentes (aunque todos tienen las capacidades del núcleo de Hadoop) y no todos los componentes en una distribución particular son compatibles con otras distribuciones, ver Tabla 11.

Proyecto/Distribución	**Cloudera**	**MapR**	**Hortonworks**
Hadoop/MapReduce	Sí	Sí	Sí
HDFS	Sí	Sí y además es compatible con NFS	Sí

Apache Pig	Sí	Sí	Sí
Apache Hive	Sí	Sí	Sí
Apache Hbase	Sí	Sí	Sí
Apache ZooKeeper	Sí	Sí	Sí
Apache Flume	Sí	Sí	Opcional
Apache Oozie	Sí	Sí	Opcional
Apache Mahout	Sí	Sí	Opcional
Apache Sqoop	Sí	Sí	Opcional
Apache Avro	Sí	Sí	No
Hue	Sí	MapR Control System	No
Cascading	No	Sí	No
Apache Lucene	No	No	No
Apache Ambari	No, soporta Cloudera Manager	No	Sí

Tabla 11: Proyectos que contienen las diferentes distribuciones (Sicular, 2012).

El criterio para seleccionar la distribución más apropiada puede pronunciarse mediante el siguiente conjunto de preguntas expresadas por (Dirk deRoos and Melnyk, 2014):

- ¿Qué se quiere lograr con Hadoop?
- ¿Cómo se puede usar Hadoop para ganar visión comercial?
- ¿Qué problemas comerciales o de negocio se quieren resolver?
- ¿Qué datos se analizarán?
- ¿Se usarán componentes propietarios o de código abierto?
- ¿Es la infraestructura de Hadoop que se está considerando suficientemente flexible para todos sus casos de uso?
- ¿Qué herramientas existentes se quieren integrar con Hadoop?
- ¿Necesitan los administradores herramientas de gestión? (La distribución del núcleo de Hadoop no incluye las herramientas administrativas.)

- ¿Le permitirá la oferta que escoja migrar a un producto diferente sin obstáculos?
- ¿La distribución que se está considerando podrá satisfacer necesidades futuras en la medida en que se puedan anticipar?

2.1.2 Especificación de las máquinas del clúster

Hadoop fue diseñado para ejecutarse no necesariamente sobre el mejor *hardware*. Lo que significa que no se está unido a las ofertas más caras o a algún *hardware* propietario de un solo vendedor; más bien, se puede escoger *hardware* normalmente disponible de cualquier vendedor para construir el clúster.

"No necesariamente sobre el mejor *hardware"* no significa el más bajo de la gama. Las máquinas de este tipo poseen a menudo componentes baratos que tienen una tasa de fracaso más alta que las máquinas que cuestan un poco más (y todavía no están en la clase más cara). Cuando se opera sobre las decenas, centenas o miles de máquinas, los componentes baratos resultan ser una economía falsa, cuando incurre una proporción de fracaso más alta, el costo de mantenimiento es mayor. Por otro lado, no se recomiendan tampoco las máquinas más caras, pues no tienen una proporción balanceada entre el precio y el rendimiento (White, 2012).

2.1.3 Configuración del clúster de Hadoop

Cada componente en Hadoop es configurado usando archivos xml. Las propiedades comunes están en *core-site.xml*, las propiedades de HDFS están en *hdfs-site.xml* y las de MapReduce en *mapred-site.xml*. En Hadoop 2.0 y posteriores, MapReduce se ejecuta sobre YARN y existe un archivo de la configuración adicional llamado *yarn-site.xml* para este componente. Todos los archivos de la configuración deben estar en el subdirectorio *etc/Hadoop*.

Según (White, 2012), Hadoop puede ejecutarse en tres modos:

- Modo autónomo (o local): no hay procesos en segundo plano y todo se ejecuta en un solo JVM (Java Virtual Machine). El modo autónomo es conveniente para los programas de MapReduce durante el desarrollo, ya que son fáciles de probar y ponerlos a punto. Este es el modo que Hadoop se encuentra por defecto.

- Modo pseudo-distribuido (nodo único): el despliegue de Hadoop en un solo nodo se refiere a la ejecución de Hadoop en modo pseudo-distribuido, donde todos los servicios de Hadoop, incluyendo los servicios maestros y esclavos, se ejecutan en una sola computadora. Este tipo de despliegue es útil para probar rápidamente aplicaciones que estén desarrollándose sin tener que preocuparse por usar los recursos de un clúster que alguien más podría necesitar. También es una manera conveniente de experimentar con Hadoop, ya que no siempre se cuenta con un clúster.

- Modo totalmente distribuido (un clúster de nodos): un despliegue de Hadoop dónde los servicios maestros y esclavos de Hadoop se ejecutan en un clúster de computadoras se conoce como el modo totalmente distribuido. Éste es un modo apropiado para clústeres de producción y clústeres de desarrollo. Una distinción puede hacerse aquí: un clúster de desarrollo normalmente tiene un pequeño número de nodos y se usa para crear prototipos de los trabajos que se ejecutarán en el futuro en un clúster de producción.

Para ejecutar Hadoop en un modo particular, se necesita hacer dos cosas: poner las propiedades convenientes y ejecutar los procesos de Hadoop. La Tabla 12 muestra el conjunto mínimo de propiedades para la configuración de cada modo. En el modo autónomo, se usan el sistema del archivos local y el proceso local encargado de ejecutar los trabajos de MapReduce (*JobTracker*), mientras que en los modos distribuidos se utilizan el HDFS y MapReduce (o YARN, para el caso de Hadoop 2.x).

Componente	**Propiedad**	**Autónomo (local)**	**Pseudo distribuido**	**Totalmente distribuido**
Common	fs.default.name	file:/// (default)	hdfs://localhost/	hdfs://name node/
HDFS	dfs.replication	N/A	1	3 (por defecto)
MapReduce	mapr.jobtracker	local	localhost:8021	Jobtracker:8021
YARN	yarn.resourceman ager.address	N/A	localhost:8032	Resourcemanager:8032

Tabla 12: Principales propiedades de configuración para los diferentes modos (White, 2012).

2.1.4 Instalación de Hadoop en la máquina virtual

Se creó una máquina virtual con el sistema operativo Lubunto 14.04. Para utilizar Hadoop en dicha máquina virtual se tomaron las siguientes decisiones:

Distribución escogida: como tal no se escogió ninguna distribución sino que se instaló el núcleo de Hadoop 2.6.0 sin ninguna otra aplicación, pues solo se utiliza MapReduce

Como parte de la instalación de Hadoop se añadieron variables de entorno al archivo *.bashrc* como se muestra en la Figura 17:

```
#HADOOP VARIABLES START

export HADOOP_INSTALL=/home/hadoop/hadoop/2.6/

export PATH=$PATH:$HADOOP_INSTALL/bin

export PATH=$PATH:$HADOOP_INSTALL/sbin

export HADOOP_MAPRED_HOME=$HADOOP_INSTALL

export HADOOP_COMMON_HOME=$HADOOP_INSTALL

export HADOOP_HDFS_HOME=$HADOOP_INSTALL

export YARN_HOME=$HADOOP_INSTALL

export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_INSTALL/lib/native

export HADOOP_OPTS="-Djava.library.path=$HADOOP_INSTALL/lib"

#HADOOP VARIABLES END
```

Figura 17: Variables de entorno para la instalación de Hadoop.

Computadoras que forman parte del clúster de Hadoop: la Tabla 13 muestra la descripción de la computadora donde se ejecutó la máquina virtual.

Aspecto	**Descripción**
Procesador	Core 2 Duo 2-3.0GHz CPUs
Memoria	1.2 GB ECC RAM (pertenece a la máquina virtual)
Almacenamiento	30 GB (pertenece a la máquina virtual)

Tabla 13: Descripción del nodo.

Modo de configuración del clúster escogido: se decide configurar el modo pseudo-distribuido con un único nodo. Según (White, 2012), para cumplir este objetivo los archivos de configuración se crean con el contenido de la Figura 18 y se guardan en el

directorio *etc/Hadoop* (también existe la posibilidad de almacenarlos en cualquier directorio, pero hay que ejecutar los procesos de Hadoop con la opción *–config <dirección del directorio que contiene los archivos de configuración>*)

```
<?xml version="1.0"?>
<!-- core-site.xml -->
<configuration>
  <property>
    <name>fs.default.name</name>
    <value>hdfs://localhost/</value>
  </property>
</configuration>

<?xml version="1.0"?>
<!-- hdfs-site.xml -->
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
</configuration>

<?xml version="1.0"?>
<!-- mapred-site.xml -->
<configuration>
  <property>
    <name>mapred.job.tracker</name>
    <value>localhost:8021</value>
  </property>
</configuration>
```

Figura 18: Configuración de Hadoop para el modo pseudo-distribuido (White, 2012).

Como se está ejecutando YARN, se utiliza el archivo *yarn-site.xml* con configuración de la Figura 19:

```
<?xml version="1.0"?>
<!-- yarn-site.xml -->
<configuration>
 <property>
  <name>yarn.resourcemanager.address</name>
  <value>localhost:8032</value>
 </property>
 <property>
  <name>yarn.nodemanager.aux-services</name>
  <value>mapreduce.shuffle</value>
 </property>
</configuration>
```

Figura 19: Configuración de YARN para el modo pseudo-distribuido (White, 2012).

Según (White, 2012), en el modo pseudo-distribuido, se necesita iniciar los procesos de Hadoop y para hacerlo se requiere la instalación de SSH[3]. Hadoop no distingue entre el modo pseudo-distribuido y el modo totalmente distribuido, solamente ejecuta los procesos que lo inician en un conjunto de nodos del clúster usando SSH. El modo pseudo-distribuido es solo un caso especial del modo totalmente distribuido en el cual no se tiene un conjunto de nodos para recrear la arquitectura maestro-esclavo de Hadoop sino que se tiene un nodo para hacerlo.

Primeramente se instala SSH de la siguiente manera (White, 2012):

% sudo apt-get install ssh

Luego, para habilitar nombre de usuario y contraseña, se genera una nueva llave SSH con una frase de contraseña vacía (White, 2012):

% ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa

% cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys

Para comprobar lo realizado hasta el momento, se utiliza el comando (White, 2012):

% ssh localhost

Antes de que los procesos de Hadoop sean ejecutados, se necesita formatear la instalación de HDFS. "El proceso de formateo origina un sistema de archivos vacío al crear los

[3] **SSH** (**S**ecure **SH**ell, en español: intérprete de órdenes segura) es el nombre de un protocolo y del programa que lo implementa, y sirve para acceder a máquinas remotas a través de una red.

directorios de almacenamiento y una versión inicial de la estructura de datos de los *NameNodes*. Los *DataNodes* no se involucran en el proceso inicial de formateo, pues los *NameNode* gestionan todos los metadatos del sistema de archivos y los *DataNode* se pueden agregar o eliminar del clúster dinámicamente. Por la misma razón no se necesita determinar inicialmente el tamaño del sistema de archivos a crear, pues esto es determinado por los *DataNodes* en el clúster (que pueden ir creciendo en número a medida que el usuario lo necesite y esté dentro de sus posibilidades) después de que el sistema de archivos sea formateado" (White, 2012).

Formatear HDFS es una operación rápida. Para ello se utiliza el comando (White, 2012):

% hadoop namenode –format

2.2 Casos de uso más frecuentes en la interacción Hadoop/MongoDB

Según la página oficial de MongoDB (2015), algunos de los casos de uso más frecuentes son:

- Agregación de lotes (ver Figura 20): en varios escenarios la funcionalidad de agregación proporcionada por MongoDB es suficiente para el análisis de sus datos. Sin embargo, en ciertos casos, puede ser necesario una agregación de datos significativamente más compleja. Aquí es donde Hadoop puede proporcionar un marco de trabajo de gran alcance para análisis complejos.

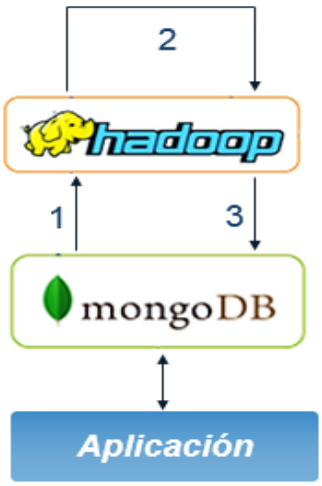

Figura 20: Ecosistema MongoDB/Hadoop utilizando Agregación de Lotes.

- Almacenes de datos (ver Figura 21): en un escenario típico de producción, los datos de su aplicación pueden estar presentes en varios almacenes de datos, cada uno con su propio lenguaje de consulta y funcionalidades. Para reducir la complejidad en estos escenarios, Hadoop puede ser utilizado como un almacén de datos y actuar como un depósito centralizado para los datos de las diversas fuentes.

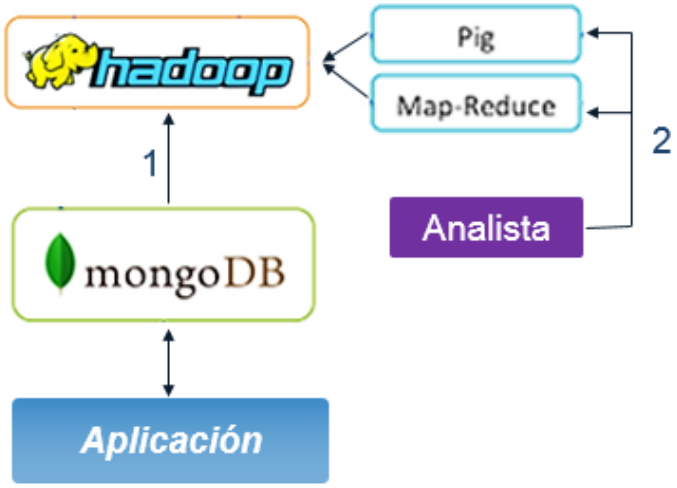

Figura 21: Ecosistema MongoDB/Hadoop utilizando Almacén de Datos.

- Extracción, transformación y carga de datos (ver Figura 22): MongoDB puede ser la fuente de datos operativa para su aplicación, pero también puede haber otras fuentes de datos que estén conteniendo los datos de su organización. En este escenario, es útil ser capaz de mover datos de una fuente de datos a otra, ya sea a partir de datos de su aplicación a otra base de datos o viceversa. Mover los datos es más complejo que simplemente usar tuberías desde un mecanismo a otro, en este proceso es donde se puede utilizar Hadoop.

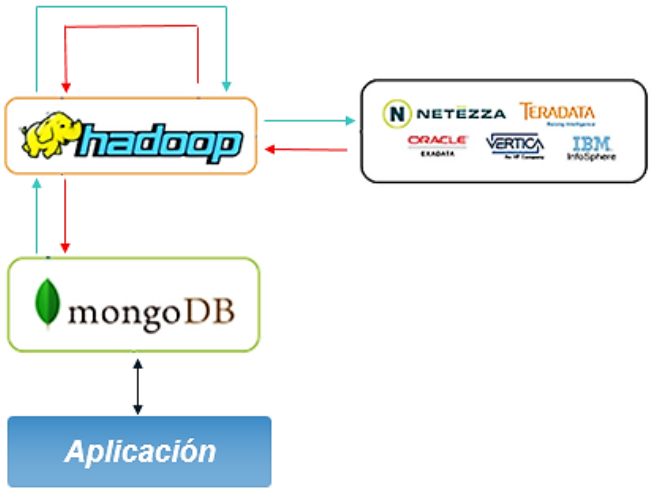

Figura 22: Ecosistema MongoDB/Hadoop utilizando ETL desde (flechas rojas) y hacia (flechas verdes) MongoDB.

2.4 Descripción del caso de estudio Automatización de Bibliotecas y Centros de Documentación (ABCD)

"Abel Parker, ex director de BIREME, realizó la presentación oficial del software ABCD en 1999. El ABCD pertenece a la familia ISIS, resultando ser una aplicación web en gestión de bibliotecas de software libre y de código abierto, que integra sus principales funciones: adquisición, catalogación, préstamos y administración de bases de datos. Su desarrollo es una iniciativa de BIREME con el apoyo del Consorcio de Universidades de Flandes en Bélgica (VLIR/UOS)" (Merino, 2013).

"Este sistema se distribuye con bases de datos MARC21, CEPAL o LILACS, permitiendo el uso de cualquier formato de catalogación y la migración de bases de datos desde WinISiS o Excel. El ABCD apunta a proporcionar, a la comunidad de usuarios de la familia ISIS, un sistema integrado para bibliotecas; incluso este software permite las exigencias de las bibliotecas universitarias, otras de las bondades del software es que ejecuta en diferentes navegadores y en cualquier computadora con Windows o UNIX/Linux" (Merino, 2013).

"Según lo expresado por BIREME, el ABCD tiene la capacidad de diseñar y adaptarse a cualquier estructura, sin restringir funcionalidades avanzadas, abriendo las puertas al manejo integrado de catálogos de bibliotecas, archivos y museos, así como a bibliotecas virtuales; en realidad este software es aplicable a cualquier tipo de bibliotecas y centros de documentación, que trabajen o no con ISIS, ya que integra todas las aplicaciones necesarias

para la automatización en línea de su trabajo, sin tener que modificar sus bases actuales, solo se deben migrar hacia ABCD" (Merino, 2013).

2.4.1 ABCD en la UCLV

La UCLV cuenta con 18 bibliotecas, las cuales trabajan directamente con el sistema ABCD. Hasta el momento, en entrevistas realizadas a expertos en el sistema se han detectado varios problemas de calidad de datos en las fuentes de ABCD. Dentro de las fuentes de datos se encuentra la base de datos NoSQL ISIS orientada a documentos denominada MARC que almacena datos semiestructurados (en la cual se centra el presente trabajo de diploma, pues es de donde se extraen los archivos *iso*, los cuales son la fuente de datos) y 10 bases de datos relacionales que manejan datos estructurados sobre: adquisiciones y copias, proveedores, sugerencias, órdenes de compra, usuarios de préstamos, objeto de préstamo, reservaciones, suspensiones y multas, transacciones de préstamo y usuarios del sistema.

Anteriormente en las bibliotecas de la UCLV se trabajaba con el sistema Quipusnet. Esta herramienta fue desarrollada por el grupo Chasqui y estudiantes de la carrera Ciencias de la Computación de la UCLV, la misma guardaba toda su información en el gestor de bases de datos Microsoft SQL Server. Por consiguiente, los datos no seguían un formato bibliográfico estandarizado, sino uno creado por especialistas del Ministerio de Educación Superior (MES).

En el año 2011 por decisión del MES se resuelve implantar ABCD en todas las bibliotecas del país y utilizar el formato bibliográfico estandarizado MARC21. Por tanto, las bibliotecas de la son sometidas a un proceso de migración de datos del antiguo sistema Quipusnet hacia ABCD. La migración no se pudo llevar a cabo, pues no existían herramientas capaces de hacerlo y el proceso era muy engorroso, por ende, se decide comenzar desde cero, o sea entrar manualmente todos los datos almacenados en Quipusnet hacia ABCD. Se ha comprobado que la migración de datos y la entrada manual como consecuencia de ello, son las causas principales que conllevaron a la existencia de problemas de calidad de datos tales como: pérdida de información, campos incompletos, valores ausentes, datos escritos incorrectamente, no estandarización de valores, por mencionar algunos.

ABCD tiene como misión: servir de soporte a la gestión bibliotecaria en las bibliotecas de los Centros de Educación Superior (CES) de Cuba, integrando en una plataforma los procesos de adquisición, catalogación, selección, procesamiento y circulación o préstamos. Además gestionar colecciones y manejar clases de usuarios.

Su visión es: ser el sistema de gestión bibliotecaria del MES que integre mediante la búsqueda federada, todos los CES de Cuba.

2.4.2 Arquitectura de ABCD

ABCD contiene varios módulos que pueden cooperar entre sí pero también pueden existir independientemente. Estos módulos se muestran en la Figura 23, en la cual se puede apreciar las relaciones jerárquicas existentes entre ellos.

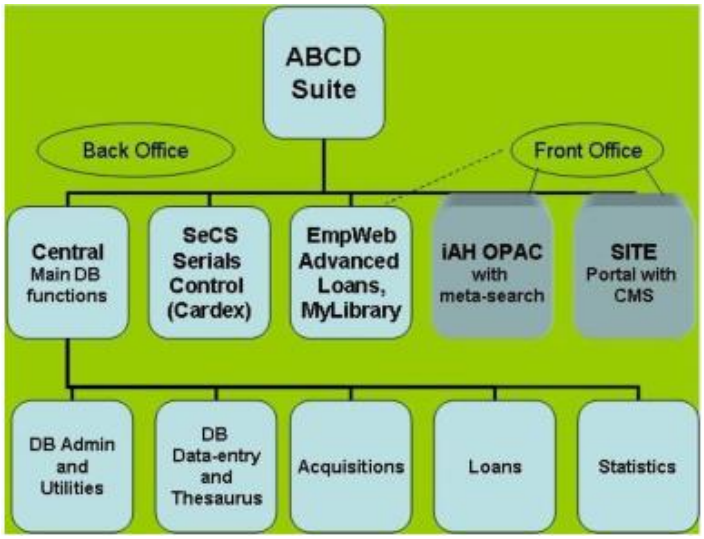

Figura 23: Relaciones jerárquicas entre los módulos de ABCD (Smet, 2009).

2.4.3 Módulos de ABCD

Según (Smet, 2009), los módulos del sistema ABCD son:

- **ABCD Central**

  El módulo central de ABCD comprende los módulos de:

  - Administración de las bases de datos.
  - Catalogación.
  - Adquisiciones.
  - Circulación, Préstamos y Estadística.

- Módulo de gestión de tesauro.
- Importación y exportación de servicios, impresión y herramientas de bases de datos.

Este módulo central representa el Back Office del sistema ABCD, los usuarios no acceden directamente a este tipo de servicio que ofrece el sistema. Cualquier estructura de base de datos ISIS puede ser definida y administrada, con registros de 1 Mb y 4 Gb como máximo en su tamaño. En comparación con la tecnología ISIS normal (con 30 caracteres), se utilizan llaves de indexación de 60 caracteres en las cuales existe mucho más control de las funciones disponibles sobre los datos. Además toda la interacción del sistema está basada en tecnología WEB lo que permite código HTML, cadenas de texto para indexación de texto completo, hipervínculos a páginas de ayuda, entre otras.

- **ABCD OPAC (iAH)**

  La interface de búsqueda pública (OPAC) es una versión adaptada de la interfaz Avanzada de Información para la Salud (iAH) de BIREME. Esto permite búsquedas por metadatos, no solo de catálogos locales, sino también de muchos otros recursos de información.

  La interfaz iAH desarrollada por BIREME está siendo actualizada a iAHx, asegurándose de que se alineará perfectamente con los conceptos modernos y técnicas de recuperación de información (por ejemplo, el agrupamiento y clasificación basada en la indexación con Lucene).

- **Sitio ABCD**

  La función de búsqueda se ofrece como parte de una página del portal para los usuarios finales, presentando el propio catálogo en un contexto de información más amplio proporcionando acceso a otros recursos de información (por ejemplo: Google o Medline) y comunicación.

- **Sistema de Control de Publicaciones Seriadas ABCD**

  Este módulo ofrece una avanzada herramienta de gestión de publicaciones seriadas (clásicas y/o electrónicas) de cualquier tipo de publicación. También pueden administrarse varios tipos de patrones de publicación por este módulo. BIREME

utiliza esta tecnología, como por ejemplo, para sus productos de Portal of Scientific Journals (Portal de Revistas Científicas) y SCAD que es el Sindicato Brasileño de Catálogos con más de 12.000 revistas de más de 50 bibliotecas.

- **ABCD Módulo Avanzado de Préstamos (EmpWeb)**

  Este módulo ofrece gestión avanzada de préstamos con algunas características extras para organizaciones más grandes y complejas.

2.4.4 Formato de los datos (MARC21)

MARC21 es una norma para el intercambio de información que permite estructurar e identificar los datos de tal forma que puedan ser reconocidos y manipulados por computadora. Este formato fue creado por un equipo de bibliotecarios de la Biblioteca del Congreso de Estados Unidos, liderados por Henriette Avram.

Según (Amed, 2012), MARC (*MAchine Readable Cataloging*) significa en español Catálogos Legibles por Máquina, donde se entiende por:

- Máquina: a cualquier computadora que soporte un sistema automatizado de administración de bibliotecas.
- Legible por máquina: al proceso mediante el cual una máquina puede leer e interpretar los datos contenidos en un registro catalográfico.

Elementos en la creación de los registros legibles en máquina (Amed, 2012):

- Elementos no catalográficos: Los números de control que los sistemas asignan a cada registro ingresado a la base de datos y que están directamente relacionados con títulos y en general elementos catalográficos del ítem descrito.
- Elementos catalográficos: Todos los que contenga el ítem que se describe (título, edición, pie de imprenta, características físicas, serie, número normalizado, notas). Se incluyen además el número de clasificación, entrada principal y registro, entre otros.

2.5 Conclusiones parciales

En este capítulo se configuró una máquina virtual con la instalación de Hadoop para el procesamiento de los datos y MongoDB para el almacenamiento en donde la máxima

potencialidad está en la interacción entre ambos. Además, se explicaron los pasos a seguir para instalar y configurar satisfactoriamente la versión Hadoop 2.6.0. Por último, se describió el caso de estudio ABCD, del cual se determinó trabajar con el módulo de Catalogación y la base de datos MARC, la cual posee los registros bibliográficos en el formato estandarizado MARC21.

CAPÍTULO 3: IMPLEMENTACIÓN DE LA MÉTRICA DE CALIDAD DE DATOS "COMPLETITUD".

En este capítulo se diseña una arquitectura para la medición de la dimensión completitud, en la cual se muestran las tecnologías *big data* utilizadas en el caso de estudio ABCD. Se diseña también la base de datos en MongoDB para el almacenamiento de los registros bibliográficos. Se implementa una herramienta para el trabajo con la fuente de datos y dos trabajos utilizando MapReduce. De estos últimos se presentan los pasos a seguir para su correcta ejecución y los resultados que brindan sobre la fuente de datos.

3.1 Arquitectura para la medición de la dimensión completitud

Para el proceso de medición de una dimensión de la calidad de los datos se diseña una arquitectura basada en la fuente de los datos y en las tecnologías *big data* utilizadas. En sentido general, la arquitectura cuenta con un total de cinco funciones básicas, de las cuales la primera denominada extracción depende de los administradores responsables de ABCD. Esta función consiste en extraer los archivos, con extensión *iso*, generados por ABCD a partir de la base de datos ISIS, los cuales contienen los registros bibliográficos en el formato semiestructurado MARC21.

Para realizar las funciones de importar y exportar los archivos *iso* a una base de datos en MongoDB, así como la de visualización parcial se implementa una herramienta llamada *Iso_MongoDB*. La misma se desarrolla sobre el lenguaje de programación *java*.

Para la función de procesar los datos en MongoDB se implementan dos trabajos MapReduce, de los cuales el primero mide la completitud de cada registro y el segundo brinda una comparación por intervalos de completitud entre los archivos *iso*.

En la Figura 24 se presenta el diseño de la arquitectura propuesta.

Figura 24: Arquitectura para la medición de la dimensión completitud.

3.1.1 Estructura de los registros MARC21

En el proceso de extracción de los datos bibliográficos se generan los archivos *iso*, los cuales contienen los registros bibliográficos.

Según (García, 2008), un registro bibliográfico en formato MARC21 se compone de:

- Cabecera: campo fijo que comprende las primeras 24 posiciones (00-23) de cada registro y suministra información para el procesamiento del registro.

  Posiciones importantes de los caracteres:

  - 00-04 (Longitud del registro): cadena de cinco caracteres, generada por el sistema que especifica la longitud del registro completo. El número se justifica a la derecha y toda posición no utilizada contiene un número cero.
  - 12-16 (Dirección base para los datos): cadena numérica de cinco caracteres generada por el sistema que indica la primera posición del carácter del primer campo variable de control dentro del registro. El número se justifica a la derecha y cada posición no utilizada contiene un número cero.

- Directorios: índice generado por sistema de la localización de los campos de datos y control variables dentro de un registro. El directorio aparece inmediatamente a continuación de la cabecera en la posición 24 y consiste en una serie de entradas de

longitud fija (12 caracteres) que indican la etiqueta, la longitud y la posición del carácter inicial de cada campo variable.

Posiciones de los caracteres:

- 00-02 (Etiqueta): tres caracteres numéricos o alfabéticos (mayúsculas o minúsculas, pero no ambas) que identifican un campo asociado.
- 03-06 (Longitud de campo): cuatro caracteres numéricos que indican la longitud de campo, incluyendo los indicadores, los códigos de subcampos, los datos y el elemento terminador de campo. El número se justifica a la derecha y las posiciones no utilizadas contienen un cero.
- 07-11 (Posición del carácter inicial): cinco caracteres numéricos que indican la posición del carácter inicial del campo en relación al inicio de los datos de la cabecera del registro (carácter 12 al 16). El número se justifica a la derecha y las posiciones no utilizadas contienen un cero.

- Campos: para su descripción, los elementos catalográficos han sido divididos en dos grandes grupos:
  - Campos de longitud fija (de tipo rígido): son todos aquellos elementos que, mediante la utilización de claves asignadas para su establecimiento, no ocupan más caracteres que los que les fueron asignados originalmente.
  - Campos de longitud variable (de tipo flexible): permiten el ingreso de datos de los elementos catalográficos contenidos en el ítem que se describe, independientemente del total de caracteres que tengan. Por ejemplo, se tienen dos títulos:
    - Formato MARC21 para registros bibliográficos (44 caracteres incluyendo espacios).
    - Hadoop in action (16 caracteres incluyendo espacios).

3.2 Diseño de la base de datos en MongoDB

En MongoDB se crea una base de datos llamada "*marc21*" que almacena todos los registros bibliográficos, donde dicha base de datos contiene cada archivo *iso* como una colección y cada registro como un documento.

Cada documento se configura de la siguiente manera (ver Figura 25):

- Primeramente se almacena la llave *header_AND_directories* almacenando como valor la cabecera y los directorios de los registros.
- Luego se almacenan como llave los directorios y como valor los subcampos que contiene el campo al cual hace referencia. El valor de los subcampos se colecciona en un arreglo de caracteres.

```
{
    "header_AND_directories": <cabecera y directorios>
    <directorio 1>: [
        <subcampo 1>,
                :
        <subcampo m>
    ]
    :
    <directorio n>: [
        <subcampo 1>,
                :
        <subcampo m>
    ]
}
```

Figura 25: Diseño de los documentos.

La razón por la cual se decidió almacenar los directorios y no la etiqueta del campo es que existen campos repetibles, es decir, campos que aparecen más de una vez en los registros conteniendo valores diferentes. En MongoDB si una llave ya existe y se desea volver a insertar, esta se sobrescribe, lo que representa un error en el diseño de los documentos. Se garantiza el uso de los directorios como llaves porque, aunque se repitan los campos, estos siempre serán diferentes. En la Figura 26 se observa un ejemplo de registro bibliográfico.

Figura 26: Ejemplo de un registro en MongoDB utilizando la herramienta Robomongo.

3.3 Herramienta Iso_MongoDB

El propósito de esta herramienta es la importación y exportación de las fuentes de datos hacia la base de datos MongoDB, para lo cual se utiliza la biblioteca *mongo-java-driver-3.0.1* que proporciona la compañía MongoDB, para los desarrolladores de *java* y realizar una visualización parcial de las mismas.

3.3.1 Diagrama de clases

Para la diseño de la herramienta se implementaron cinco clases, además de una clase Frame que contiene la interfaz gráfica. Como se muestra en la Figura 27, las clases *Iso_to_mongo* y *Mongo_to_iso* son estáticas, pues no es necesario crear una instancia de ellas sino se usan los métodos que contienen. La primera se encarga de convertir los archivos *iso* en colecciones de documentos, en las cuales cada documento es un registro bibliográfico, e insertarlas en MongoDB. La segunda convierte las colecciones de documentos en archivos *iso* nuevamente. Para la función de visualización se formalizaron tres clases *Out_fields, Record y Extract_records_to_txt*. La primera se utiliza para almacenar los subcampos que no pertenecen al MARC conciso. La segunda almacena un registro bibliográfico haciendo

corresponder en *header* la cabecera, en *directories* los directorios, en *fields* los subcampos pertenecientes al MARC conciso y una lista de *Out_fields*. La última se encarga de realizar la visualización a partir de la extracción de los diferentes componentes de los registros.

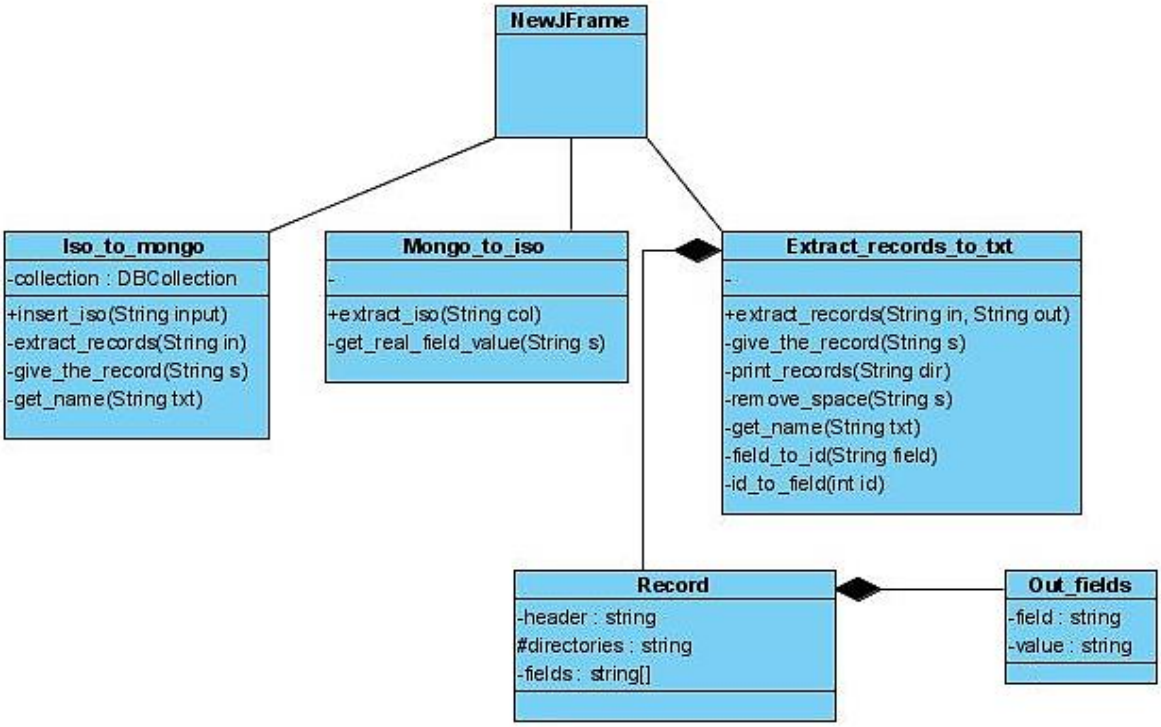

Figura 27: Diagrama de clases de la herramienta Iso_MongoDB.

3.3.2 Interfaz gráfica.

En la Figura 28 se aprecia la interfaz gráfica de la herramienta donde se puede apreciar sus tres funcionalidades. La herramienta posee un ambiente interactivo en el que es importante la participación del encargado de la base de datos en MongoDB para ejecutar el paso de medición que se esté llevando a cabo.

Figura 28: Herramienta Iso_MongoDB.

3.3.3 Importar datos hacia MongoDB

Los archivos *iso* almacenan los registros bibliográficos en formato MARC21 para la catalogación de documentos de manera automatizada.

Para extraer los registros se implementaron los siguientes pasos:

1. Conectarse a MongoDB.
2. Extraer la base de datos y la colección sobra la que se trabajará.
3. Inicializar variables.
4. Para cada línea del archivo *iso*.
    a. Mientras exista más de un registro en la línea.
        i. Caso 1: La longitud del registro es igual que la longitud de la línea.
            1. Extraer cabecera, directorios y campos.
            2. Crear el documento del registro.

3. Almacenarlo en la colección.
4. Regresar al paso 4.

ii. Caso 2: La longitud del registro es menor que la longitud de la línea.

1. Extraer cabecera, directorios y campos del primer registro.
2. Crear el documento del registro.
3. Almacenarlo en la colección.
4. Actualizar las variables.
5. Regresar al paso 4-a.

iii. Caso 3: La longitud del registro es mayor que la longitud de la línea.

1. Actualizar variables.
2. Regresar al paso 4.

3.3.4 Exportar los datos desde MongoDB

Para la exportación de los documentos a un archivo *iso* se implementaron los siguientes pasos:

1. Conectarse a MongoDB.
2. Extraer la base de datos y la colección sobra la que se trabajará.
3. Crear un archivo con extensión *iso*.
4. Para cada documento en la colección:
    a. Extraer cabecera y directorios.
    b. Para cada directorio:
        i. Extraer los valores de sus subcampos.
        ii. Generar los campos con los valores de los subcampos.
    c. Insertar cabecera, directorios y los campos en el archivo *iso* creado.

3.3.5 Visualización parcial

La visualización parcial es similar a la importación de archivos *iso* a MongoDB. Se diferencia en la salida de los datos y en esta función se especifica si los campos de los registros pertenecen o no al MARC conciso.

1. Crear archivo texto.
2. Inicializar variables.
3. Para cada línea del archivo *iso*.
    a. Mientras exista más de un registro en la línea.
        i. Caso 1: La longitud del registro es igual que la longitud de la línea.
            1. Extraer cabecera, directorios, campos que pertenecen al MARC conciso y los que no.
            2. Escribirlos en el archivo texto.
            3. Regresar al paso 4.
        ii. Caso 2: La longitud del registro es menor que la longitud de la línea.
            1. Extraer cabecera, directorios, campos que pertenecen al MARC conciso y los que no.
            2. Escribirlos en el archivo texto.
            3. Actualizar las variables.
            4. Regresar al paso 4-a.
        iii. Caso 3: La longitud del registro es mayor que la longitud de la línea.
            1. Actualizar variables.
            2. Regresar al paso 4.

Como resultado de esta función se ofrece el siguiente ejemplo. Dado el registro bibliográfico en formato MARC21 de la Figura 29, su visualización parcial se muestra en dos ilustraciones (ver Figura 30 y Figura 31).

```
00625000000000241000450098000300000000500020003000600020003200700 0
20003401700020003601800020003800800410004000100060008102000160008 7
04100070010304300070011010000250011724500730014226000440021530000 2
10025944000360028065000230031665000440033 9#^d20120913 05:54:37
^omosvaldo#n#a#m###a#110617         cu     f          0 spa d#27661#
^a959-08-0355-5#  ^aES#  ^amX#  ^aPadrón, Juan Nicolás#  ^aHéroes y padres
^buna lectura de Iliada y Odisea^cJuan Nicolás Padrón#  ^aLa Habana
^bEditorial Gente Nueva^c2001.#^a54 p.^bil., 20 cm.#
^a(Colección temas de periodismo)#  ^aLITERATURA CLASICA#
^aLITERATURA GRIEGA - ESTUDIO Y ENSEÑANZA##
```

Figura 29: Registro en formato MARC 21.

Cabecera:
00625000000000241000450

Directorios:

980003000000	008004100040	245007300142
005000200030	001000600081	260004400215
006000200032	020001600087	300002100259
007000200034	041000700103	440003600280
017000200036	043000700110	650002300316
018000200038	100002500117	650004400339

Figura 30: Visualización parcial de la cabecera y los directorios.

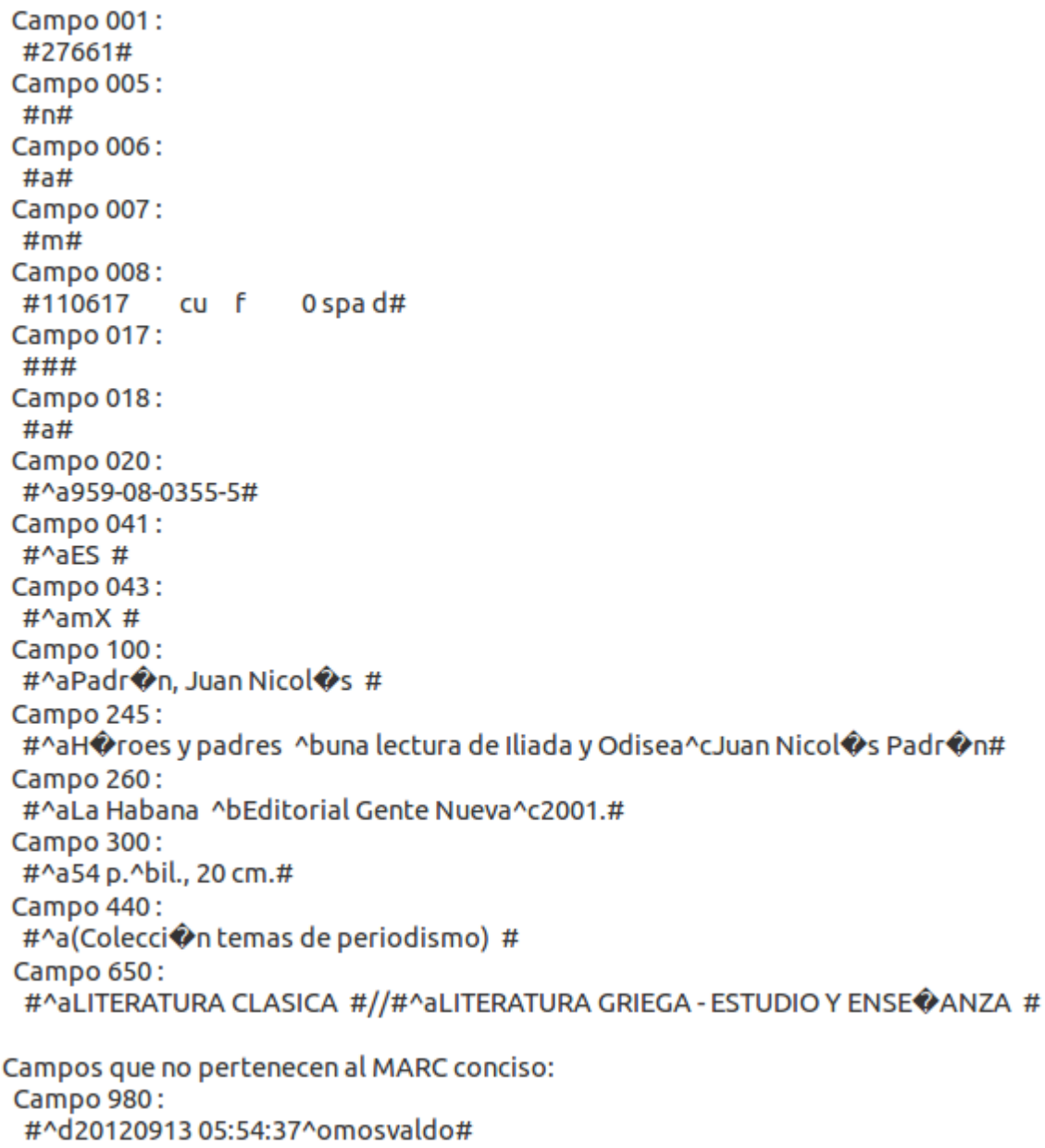

Figura 31: Visualización parcial de las campos del registro bibliográficos.

3.3 Procesamiento con Hadoop

Para el procesamiento con Hadoop se utilizan las bibliotecas *Hadoop-core-1.1.2*, *mongo-Hadoop-core-1.3.2* y *mongo-java-driver-3.0.1*. Las mismas se ubicaron en la aplicación y en la carpeta *share/Hadoop/common* dentro del marco de trabajo. Si no están en este último directorio, los nodos del clúster Hadoop no pueden tener acceso a las mismas. De esta manera se ahorra el trabajo de ponerlas en cada nodo.

3.3.1 Métrica de dimensión completitud

Según entrevistas realizadas al administrador de ABCD una de las dimensiones más importantes es la completitud. Por tanto, fue la dimensión escogida para la medición de la calidad de datos de los registros bibliográficos. En (Batini and Scannapieco, 2006) se define una métrica de calidad para evaluar la dimensión completitud para datos relacionales mediante la siguiente fórmula: $\mathbf{C}(\mathbf{r}) = |\mathbf{r}|/|\mathbf{ref}(\mathbf{r})|$, donde r es una relación, **|r|** es la cantidad de valores almacenados en la relación exceptuando los valores nulos y **ref(r)** es la cantidad real de valores en dicha relación, incluyendo los nulos.

Como la base de datos que se está trabajando no es relacional, los valores de la métrica de completitud se modifican y se obtuvo la completitud por registro definida como: $\boldsymbol{C(registro)} = \frac{(\boldsymbol{conteo} * \mathbf{100})}{\mathbf{205}}$, donde conteo es la cantidad de campos que pertenecen al MARC conciso del registro analizado y 205 es la cantidad de campos total del MARC conciso. Se multiplica por 100 para obtener el valor de completitud en por ciento para una mejor comprensión.

3.3.1.1 Implementación de la métrica

La completitud por cada registro bibliográfico se implementó usando el modelo de programación Hadoop/MapReduce. La cual se dividió en dos clases fundamentales: *Hadoop_MongoDB* y *Completness_map*.

Clase Hadoop_MongoDB: es la encargada de configurar y ejecutar el trabajo MapReduce.

Procedimiento:

1. Conectarse a la base de datos en MongoDB.
2. Configuración del trabajo MapReduce.
3. Ejecutar el trabajo.
4. Si el trabajo terminó correctamente salida con éxito sino salida con error.

Clase Completness_map: es la encargada de emitir una lista de los títulos de los registros junto al por ciento de los campos de los mismos que pertenecen al MARC conciso.

Definición:

map (*llave, documento****) → (****id, por_ciento****)***

Procedimiento:

Entrada: Se toma como entrada cada registro almacenado, como un documento, en una determinada colección en MongoDB.

1. Extraer cabecera y directorios del registro.
2. Para cada campo del registro.
    a. Chequear cuántos campos del MARC conciso posee.
    b. Extraer del campo "245" el subcampo "a", si lo posee (esto es el título del registro).
    c. Calcular la completitud por registro.
    d. Si tiene título devolverlo como llave, sino la cabecera y como valor el por ciento de completitud del registro.

3.3.1.2 Pasos para ejecutar la métrica implementada

Para realizar una correcta ejecución de la métrica completitud es necesario seguir una serie de pasos, para ello se utilizó el archivo "Marc.iso":

1. Acceder a la carpeta *sbin* donde se encuentra el marco de trabajo Hadoop 2.6.0.
   % cd hadoop/2.6/sbin
2. Ejecutar los procesos de HDFS y YARN (ver Figura 32).
   % ./star-all.sh

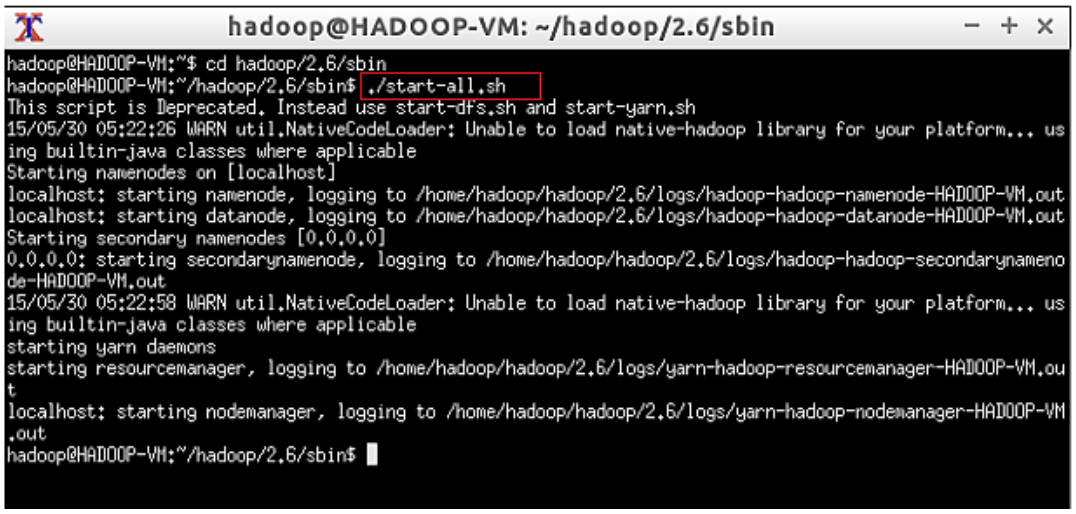

Figura 32: Comando para ejecutar los procesos de Hadoop.

3. Escribir el comando para ejecutar el archivo jar para la medición de la calidad de datos en la dimensión de completitud por registros (ver Figura 33).

   % hadoop jar /home/hadoop/hadoop/dist/Hadoop_MongoDB.jar Marc /output

   Donde:

   - **hadoop jar** es el comando para ejecutar los archivos jar.
   - **/home/hadoop/hadoop/dist/Hadoop_MongoDB.jar** es la ubicación del jar en el sistema de archivos local.
   - **Marc** es la colección que se desea analizar.
   - **/output** es la ubicación del archivo de salida en HDFS.

Figura 33: Comando para ejecutar los trabajos MapReduce.

Cuando Hadoop ejecuta un trabajo se va mostrando el por ciento de acabado de las funciones *map* y *reduce* (ver Figura 34).

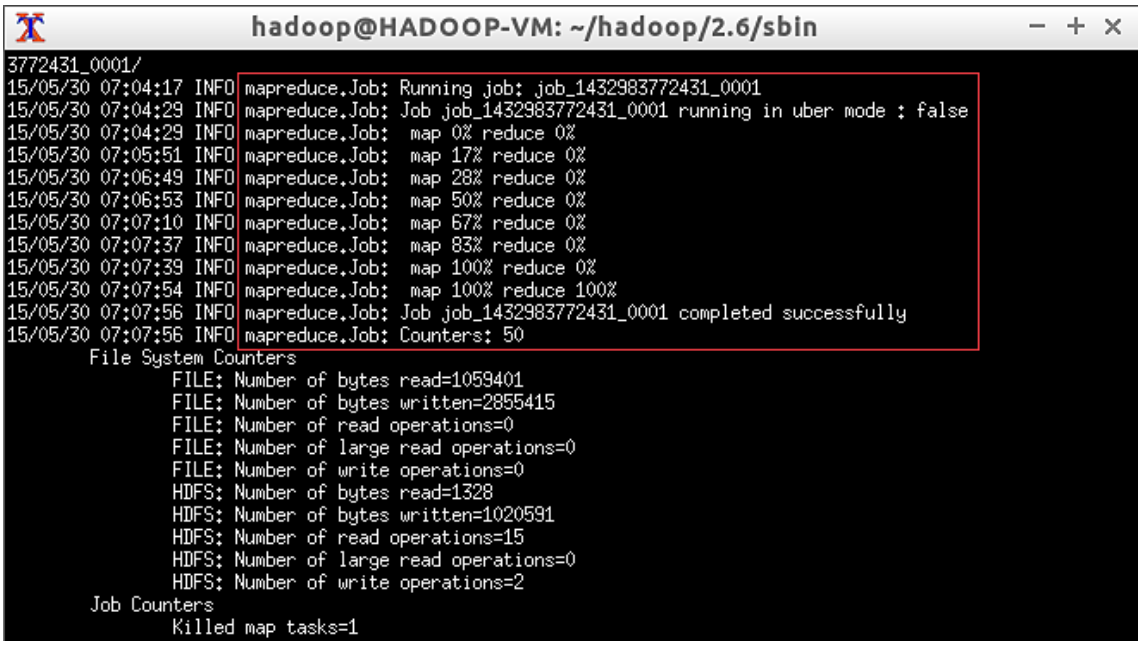

Figura 34: Proceso de ejecución de las funciones map y reduce.

4. Para visualizar los resultados se ejecuta el comando:

   % hadoop fs –cat /output/part-r-00000

   Donde part-r-00000 es el archivo creado por Hadoop/MapReduce en el archivo de salida (ver Figura 35). Hadoop por defecto asigna su nombre.

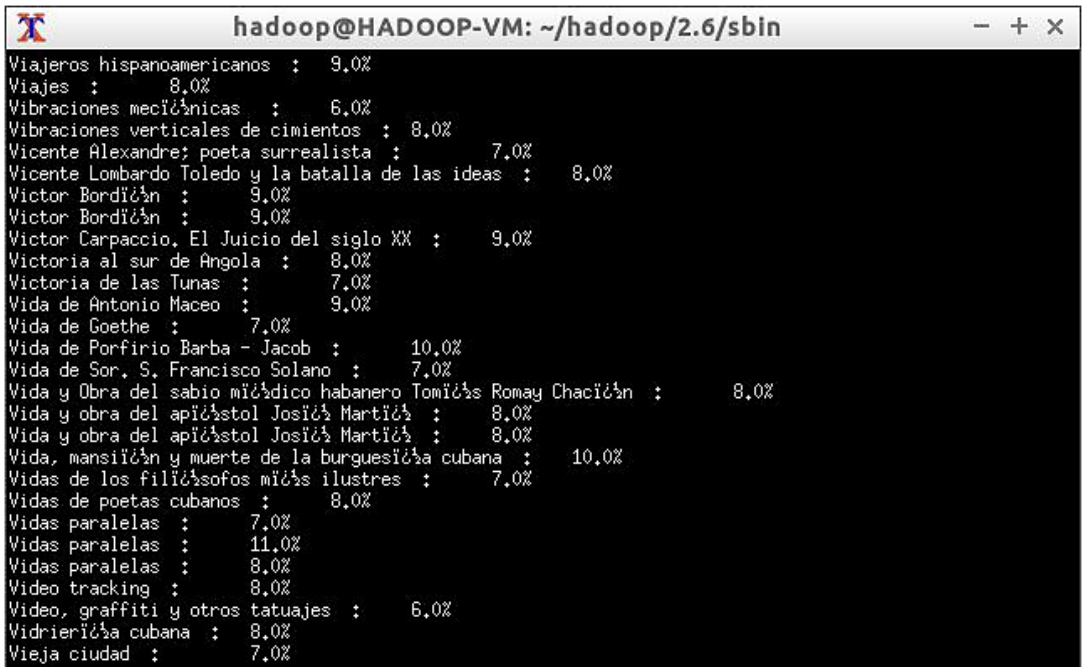

Figura 35: Resultados de ejecutar la métrica implementada en Hadoop.

3.3.2 Trabajo MapReduce para comparar los archivos iso por intervalos de completitud

Este trabajo MapReduce se creó con el objetivo de comparar los diferentes archivos iso a través de la métrica implementada en cinco intervalos de completitud, las cuales son: de 0 a 5%, de 5 a 10%, de 10 a 15%, de 15 a 20% y más de 20%. Para cumplir este objetivo se crearon tres clases: *Hadoop_MongoDB_count*, *Completness_count_map* y *Completness_count _map*.

Clase Hadoop_MongoDB_count: es la encargada de configurar y ejecutar el trabajo MapReduce.

Procedimiento:

5. Conectarse a la base de datos en MongoDB.
6. Configuración del trabajo MapReduce.
7. Ejecutar el trabajo.
8. Si el trabajo terminó correctamente salida con éxito sino salida con error.

Clase Completness_map_count: es la encargada de emitir a qué intervalo pertenece el registro que se está analizando y contarlo.

Definición:

map(*llave, documento****)*** → ***(****intervalo, 1****)***, donde intervalo es dada por el por ciento de completitud del registro.

Procedimiento:

Entrada: Se toma como entrada cada registro almacenado, como un documento, en una determinada colección en MongoDB.

1. Extraer directorios del registro.
2. Para cada campo del registro.
    a. Chequear cuántos campos del MARC conciso posee.
    b. Calcular la completitud por registro.
    c. Devolver como llave la intervalo y como valor 1.

Clase Completness_reduce_count: es la encargada de contar la cantidad de intervalos de completitud emitidas por la función *map*.

Definición:

reduce(*intervalo, lista[1,…,1]****)*** → ***(****intervalo, suma****)***, donde lista[1,…,1] es generada por el proceso de *Shuffle and Sort* de Hadoop, de manera que las salidas de *map* con la misma llave (o con el mismo intervalo) son combinadas en una lista para formar un solo registro de entrada para la reducción. Suma es la sumatoria de dicha lista.

Procedimiento:

Entrada: Se toma como entrada el intervalo y la lista asociada a ella.

1. Realizar una sumatoria de la lista.
2. Devolver como llave el intervalo y como valor la sumatoria.

3.3.2.2 Comparación de los archivos iso por los intervalos de completitud

Del caso de estudio se tomaron los cuatro archivos *iso* que contenían registros bibliográficos, el resto son de registros de auditores, en formato MARC21 llamados Marc, final_marc_win2lin, marc_win2lin_corrected y recovered_marc_win2lin, los cuales contienen 18484, 27592, 27608 y 27627 registros respectivamente.

En primer lugar se almacenaron en MongoDB y se les aplicó el trabajo *Hadoop_MongoDB_count*, obteniéndose resultados en la Tabla 14 para ser comparados, a partir de los cinco intervalos creados:

Archivo *iso*	**0-5%**	**5-10%**	**10-15%**	**15-20%**	**+20%**	**Total de registros**
Marc	181	17554	745	4	0	18484
final_marc_win2lin	13	25684	1893	0	2	27592
marc_win2lin_corrected	21	25693	1892	0	2	27608
recovered_marc_win2lin	22	25710	1893	0	2	27627

Tabla 14: Tabla comparativa de los cinco intervalos.

Para un mejor análisis de la tabla se puede apreciar en Figura 36 un gráfico de barras en donde el mayor número de registros bibliográficos presentan entre un 5 a un 10% de completitud.

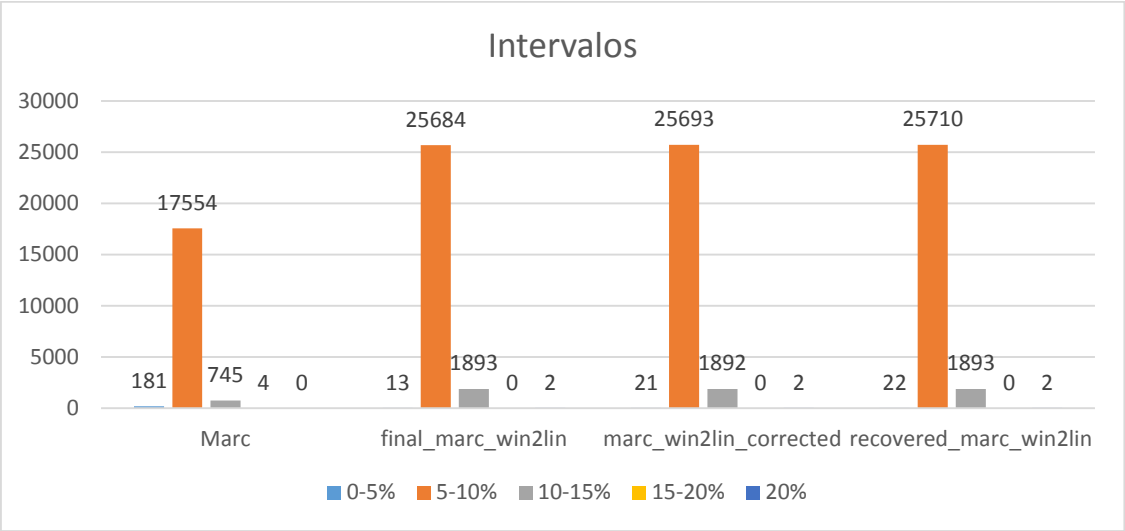

Figura 36: Análisis de los archivos iso.

Al realizar una observación más detallada (ver Figura 37) de los cuatro archivos *iso* se puede apreciar que el intervalo 5-10% se encuentra entre el 93 y 95% de completitud y el intervalo de 10-15% se registra entre el 4 y 7%, lo que puede traer como consecuencias un análisis incompleto de los registros bibliográficos, el descontento de los usuarios y disminuye la satisfacción de los empleados.

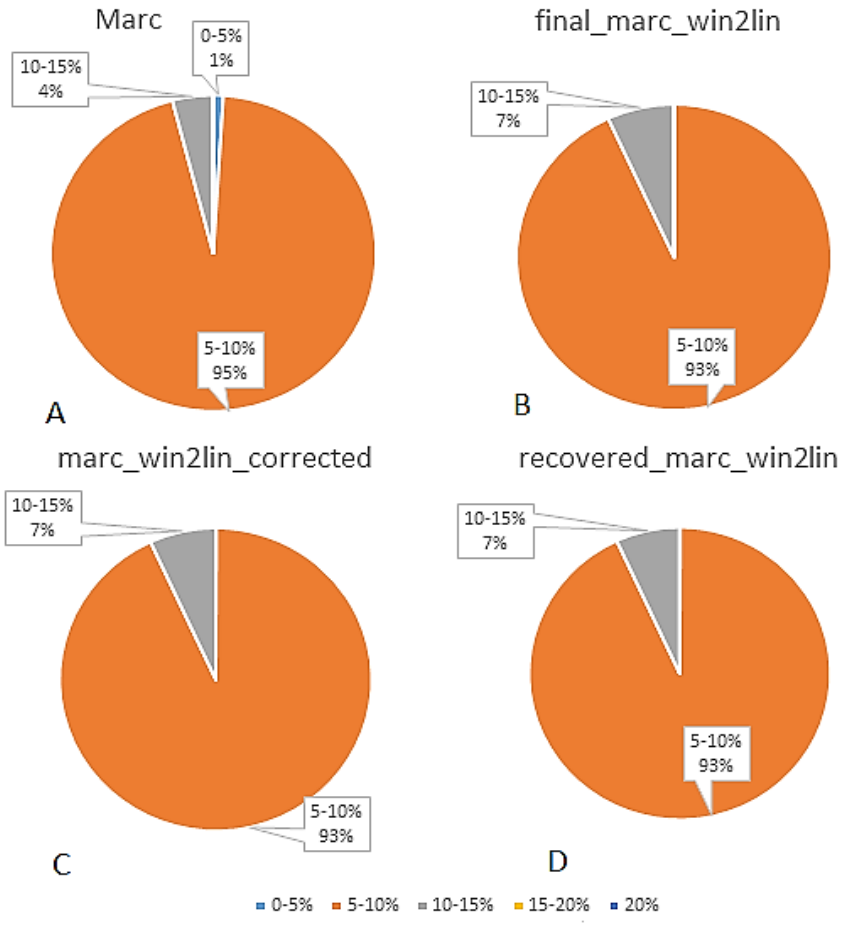

Figura 37: Observación detallada de cada archivo iso.

3.4 Conclusiones parciales

En este capítulo se diseñaron una arquitectura para la medición de la calidad de datos y una base de datos en MongoDB llamada "*marc21*" para el almacenamiento de las colecciones de documentos, los cuales son los registros bibliográficos de los archivos *iso*. Se implementó la herramienta *Iso_MongoDB*, que permite la importación, exportación de archivos *iso* hacia MongoDB y una visualización parcial de los archivos *iso*. Además se implementaron dos trabajos con el modelo de programación Hadoop/MapReduce: *Completeness*, el cual es el encargado de reflejar en un archivo los por cientos de completitud de cada registro bibliográfico de una determinada colección en MongoDB y *Completeness_count* que crea un archivo con intervalos de completitud de una colección en MongoDB.

CONCLUSIONES

En el presente trabajo de diploma:

- Se analizaron las principales características y funcionalidades de las herramientas *big data*.
- Se configuró una máquina virtual con la instalación de Hadoop y MongoDB a modo de prueba para ejecutar los trabajos MapReduce elaborados.
- Se implementó una herramienta para la importación y exportación de archivos *iso* hacia MongoDB, que permite además visualizarlos parcialmente.
- Se implementó una métrica para la evaluación de la dimensión de calidad de datos completitud en la base de datos NoSQL denominada MARC perteneciente al módulo Catalogación del sistema ABCD, utilizando Hadoop/MapReduce.

RECOMENDACIONES

- Implementar el resto de las métricas para evaluar la calidad de las dimensiones que resultaron más importantes en el procesamiento de cuestionarios aplicados a los especialistas que trabajan con ABCD.
- Configurar Hadoop y MongoDB en un clúster de computadoras.

BIBLIOGRAFÍA

2015. *Hadoop and MongoDB Use Cases* [Online]. MongoDB, Inc. Available: https://www.mongodb.com/.

ABADI, D. J., BONCZ, P. A. & HARIZOPOULOS, S. 2009. Column-oriented database systems. *Proceedings of the VLDB Endowment,* 2**,** 1664-1665.

AGRAWAL, D., BERNSTEIN, P., BERTINO, E., DAVIDSON, S., DAYAL, U., FRANKLIN, M., GEHRKE, J., HAAS, L., HALEVY, A., HAN, J., JAGADISH, H. V., LABRINIDIS, A., MADDEN, S., PAPAKONSTANTINOU, Y., PATEL, J. M., RAMAKRISHNAN, R., ROSS, K., SHAHABI, C., SUCIU, D., VAITHYANATHAN, S. & WIDOM, J. 2012. Challenges and Opportunities with Big Data.

AMED 2012. Curso de formación bibliotecaria: Manejo y aplicación del Formato MARC21 (Básico).

BATINI, C., CAPPIELLO, C., FRANCALANCI, C. & MAURINO, A. 2009. Methodologies for data quality assessment and improvement. *ACM Computing Surveys (CSUR),* 41**,** 16.

BATINI, C. & SCANNAPIECO, M. 2006. *Data quality: concepts, methodologies and techniques*, Springer.

BERMAN, J. J. 2013. *Principles of big data: preparing, sharing, and analyzing complex information*, Newnes.

BOBROWSKI, M., MARRÉ, M. & YANKELEVICH, D. 1999. Measuring data quality. *Universidad de Buenos Aires. Report***,** 99-002.

COX, M. & ELLSWORTH, D. Managing big data for scientific visualization. ACM Siggraph, 1997. 21.

CHEN, M., MAO, S. & LIU, Y. 2014. Big data: A survey. *Mobile Networks and Applications,* 19**,** 171-209.

CHEN, Y., ALSPAUGH, S. & KATZ, R. 2012. Interactive analytical processing in big data systems: A cross-industry study of mapreduce workloads. *Proceedings of the VLDB Endowment,* 5**,** 1802-1813.

CHODOROW, K. 2013. *MongoDB: The Definitive Guide, Second Edition*, O'Reilly Media, Inc.

DEAN, J. & GHEMAWAT, S. 2008. MapReduce: simplified data processing on large clusters. *Communications of the ACM,* 51**,** 107-113.

DECANDIA, G., HASTORUN, D., JAMPANI, M., KAKULAPATI, G., LAKSHMAN, A., PILCHIN, A., SIVASUBRAMANIAN, S., VOSSHALL, P. & VOGELS, W. Dynamo: amazon's highly available key-value store. ACM SIGOPS Operating Systems Review, 2007. ACM, 205-220.

DEMCHENKO, Y., NGO, C. & MEMBREY, P. 2013. Architecture Framework and Components for the Big Data Ecosystem. *Journal of System and Network Engineering***,** 1-31.

DIRK DEROOS, P. C. Z., BRUCE BROWN, RAFAEL COSS, & MELNYK, R. B. 2014. *Hadoop For Dummies*, John Wiley & Sons, Inc.

DU, C., GUO, H. & QIU, Y. 2014. Bigtable: A Distributed Storage System for Structured Data.

EBNER, K., BUHNEN, T. & URBACH, N. Think Big with Big Data: Identifying Suitable Big Data Strategies in Corporate Environments. System Sciences (HICSS), 2014 47th Hawaii International Conference on, 2014. IEEE, 3748-3757.

ECKERSON, W. W. 2002. Data quality and the bottom line. *TDWI Report, The Data Warehouse Institute*.

FAN, W. & GEERTS, F. 2012. Foundations of data quality management. *Synthesis Lectures on Data Management,* 4**,** 1-217.

FRANKS, B. 2012. *Taming the big data tidal wave: Finding opportunities in huge data streams with advanced analytics*, John Wiley & Sons.

GANTZ, J. & REINSEL, D. 2011. Extracting value from chaos. *IDC iview***,** 9-10.

GARCÍA, A. 2008. *Formato Marc 21 para registros bibliográficos*.

GEORGE, L. 2011. *HBase: the definitive guide*, " O'Reilly Media, Inc.".

HASHEM, I. A. T., YAQOOB, I., ANUAR, N. B., MOKHTAR, S., GANI, A. & KHAN, S. U. 2014. The rise of "big data" on cloud computing: Review and open research issues. *Information Systems,* 47**,** 98-115.

HOFFMAN, S. 2013. *Apache Flume: Distributed Log Collection for Hadoop*, Packt Publishing Ltd.

HOLMES, A. 2012. *Hadoop in practice*, Manning Publications Co.

LAKSHMAN, A. & MALIK, P. 2011. The Apache cassandra project. Online] http://cassandra. apache. org.

LANEY, D. 2001. *3-D Data Management: Controlling Data Volume, Velocity and Variety* [Online].

LEAVITT, N. 2010. Will NoSQL databases live up to their promise? *Computer,* 43**,** 12-14.

MANYIKA, J., CHUI, M., BROWN, B., BUGHIN, J., DOBBS, R., ROXBURGH, C., BYERS, A. H. & INSTITUTE, M. G. 2011. Big data: The next frontier for innovation, competition, and productivity.

MERINO, R. M. 2013. *Acerca del sistema ABCD para bibliotecas.* [Online].

MORENO, J. P. 2014. Una aproximación a Big Data. *Revista de Derecho de la UNED (RDUNED)***,** 471-506.

NEUBAUER, P. 2010. Graph databases, NOSQL and Neo4j.

NEUMEYER, L., ROBBINS, B., NAIR, A. & KESARI, A. S4: Distributed stream computing platform. Data Mining Workshops (ICDMW), 2010 IEEE International Conference on, 2010. IEEE, 170-177.

NUGENT, A., HALPER, F. & KAUFMAN, M. 2013. *Big data for dummies*, John Wiley & Sons.

O'LEARY, D. E. 2013. Artificial intelligence and big data. *IEEE Intelligent Systems,* 28**,** 0096-99.

OLSTON, C., REED, B., SRIVASTAVA, U., KUMAR, R. & TOMKINS, A. Pig latin: a not-so-foreign language for data processing. Proceedings of the 2008 ACM SIGMOD international conference on Management of data, 2008. ACM, 1099-1110.

OWEN, S., ANIL, R., DUNNING, T. & FRIEDMAN, E. 2011. *Mahout in action*, Manning.

PIPINO, L. L., LEE, Y. W. & WANG, R. Y. 2002. Data quality assessment. *Communications of the ACM,* 45**,** 211-218.

PORRERO, B. L. 2011. *Limpieza de datos: Reemplazo de valores ausentes y estandarización.*, Universidad Central "Marta Abreu" de Las Villas.

QUACKENBUSH, J. 2002. Microarray data normalization and transformation. *Nature genetics,* 32**,** 496-501.

RABKIN, A. & KATZ, R. H. Chukwa: A System for Reliable Large-Scale Log Collection. LISA, 2010. 1-15.

RADIA, S. & SRINIVAS, S. 2014. Hadoop 2: What's New. *Login,* 39**,** 12-15.

RAO, B., SALUIA, P., SHARMA, N., MITTAL, A. & SHARMA, S. Cloud computing for Internet of Things & sensing based applications. Sensing Technology (ICST), 2012 Sixth International Conference on, 2012. IEEE, 374-380.

REDMAN, T. C. 1998. The impact of poor data quality on the typical enterprise. *Communications of the ACM,* 41**,** 79-82.

RIJMENAM, M. V. 2015. *Why The 3V's Are Not Sufficient To Describe Big Data* [Online].

SAHA, B. & SRIVASTAVA, D. Data quality: The other face of Big Data. Data Engineering (ICDE), 2014 IEEE 30th International Conference on, 2014. IEEE, 1294-1297.

SARSfiELD, S. 2011. The butterfly effect of data quality. *The Fifth MIT Information Quality Industry Symposium*.

SCHROECK, M., SHOCKLEY, R., SMART, J., MORALES, D. R. & TUFANO, P. 2012. Analytics: el uso de big data en el mundo real. Cómo las empresas más innovadoras extraen valor de datos

inciertos. IBM® Institute for Business Value, Escuela de Negocios Saïd en la Universidad de Oxford.

SEEGER, M. & ULTRA-LARGE-SITES, S. 2009. Key-Value stores: a practical overview. *Computer Science and Media, Stuttgart*.

SICULAR, S. 2012. *The Charging Elephant* [Online].

SMET, E. D. 2009. *The abc of ABCD: the Reference Manual.*

STAROSTENKOV, V. & GRIGORCHUK, K. 2013. Hadoop Distributions: Evaluating Cloudera, Hortonworks and MapR in Micro-benchmarks and Real-world Applications. *Altoros Systems, Inc.*

STEPHEN KAISLER, FRANK ARMOUR, J. ALBERTO ESPINOSA & MONEY, W. 2013. Big Data: Issues and Challenges Moving Forward.

STRONG, D. M., LEE, Y. W. & WANG, R. Y. 1997. Data quality in context. *Communications of the ACM,* 40**,** 103-110.

SYED, A. R., GILLELA, K. & VENUGOPAL, C. 2013. The Future Revolution on Big Data. *Future,* 2.

TAYLOR, R. C. 2010. An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics. *BMC bioinformatics,* 11**,** S1.

TEE, J. 2013. Handling the four 'V's of big data: volume, velocity, variety, and veracity. *TheServerSide.com*

THUSOO, A., SARMA, J. S., JAIN, N., SHAO, Z., CHAKKA, P., ANTHONY, S., LIU, H., WYCKOFF, P. & MURTHY, R. 2009. Hive: a warehousing solution over a map-reduce framework. *Proceedings of the VLDB Endowment,* 2**,** 1626-1629.

VAVILAPALLI, V. K., MURTHY, A. C., DOUGLAS, C., AGARWAL, S., KONAR, M., EVANS, R., GRAVES, T., LOWE, J., SHAH, H. & SETH, S. Apache hadoop yarn: Yet another resource negotiator. Proceedings of the 4th annual Symposium on Cloud Computing, 2013. ACM, 5.

VENNER, J. 2009. *Pro Hadoop*.

WANG, R. Y. & STRONG, D. M. 1996. Beyond accuracy: What data quality means to data consumers. *Journal of management information systems***,** 5-33.

WHITE, T. 2012. *Hadoop: The Definitive Guide*, O'Reilly Media, Inc.

ZAHARIA, M., CHOWDHURY, M., FRANKLIN, M. J., SHENKER, S. & STOICA, I. Spark: cluster computing with working sets. Proceedings of the 2nd USENIX conference on Hot topics in cloud computing, 2010. 10-10.

ZIKOPOULOS, P., PARASURAMAN, K., DEUTSCH, T., GILES, J. & CORRIGAN, D. 2012. *Harness the Power of Big Data The IBM Big Data Platform*, McGraw Hill Professional.