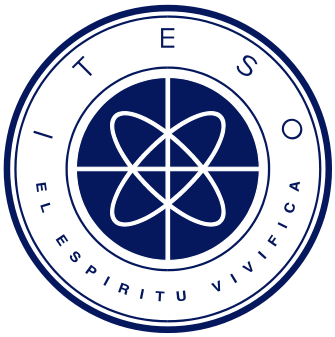
INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

Departamento de Electrónica, Sistemas e Informática

Maestría en Sistemas Computacionales



Herramienta de *Business Intelligence* para el soporte en la toma de decisiones del sector salud

Trabajo recepcional que para obtener el grado de

MAESTRO EN SISTEMAS COMPUTACIONALES

Presenta: Ing. Horacio Iturbe García
CVU: 788030

Director: Mtro. Víctor Hugo Ortega Guzmán

Tlaquepaque, Jalisco. Septiembre del 2018.

AGRADECIMIENTOS

Agradezco ampliamente al Centro Nacional de Ciencia y Tecnología (CONACYT) por la beca otorgada para cursar mis estudios de Maestría, con No. de Beca 445688 y CVU: 788030

A mi director de tesis, Mtro. Víctor Hugo Ortega Guzmán por todo su apoyo y su gran disponibilidad en todo momento.

DEDICATORIA

A mi esposa Katy:

Por estar a mi lado, por su paciencia y su apoyo incondicional.

RESUMEN

Este proyecto, se centra en el análisis de la interacción entre afecciones y medicamentos a partir de la información que es publicada por el sector salud de los años 2008 al 2015. Para ello se desarrolló un prototipo.

Para realizar el análisis de la información, se propone los datos sean manejados y almacenados como grafos. Esto nos ofrece la posibilidad de identificar, si hay alguna afección que sea la causante de otra o bien su grado de incidencia. Mediante una interfaz web, el usuario final, selecciona una o varias afecciones y medicamentos a consultar.

Los resultados son mostrados en la interfaz y se dividen en tres secciones. La primera sección ofrece información sobre los medicamentos que han sido encontrados en los conjuntos publicados por el sector salud, como prescritos a la afección seleccionada. La segunda sección, muestra las afecciones a las cuales ha sido prescrito el medicamento o los medicamentos seleccionados. Finalmente, la tercera sección muestra otras afecciones en un nivel de profundidad uno (los vecinos cercanos), que están relacionadas con la afección o afecciones seleccionadas.

La solución se implementó utilizando únicamente herramientas de código abierto.

LISTA DE TABLAS

LISTA DE ACRÓNIMOS Y ABREVIATURAS

DGIS		Dirección General de Información en Salud
IMSS		Instituto Mexicano del Seguro Social
INEGI		Instituto Nacional de Estadística y Geografía
TALEND		*Talend Open Studio for Big Data*
HDFS		*Hadoop Distributed File System*
SQL		*Structured Query Language*
BD		*Big Data*
ML		*Machine Learning*
ETL		*Extract Transform and Load*
NoSQL		*Not Only SQL*
ACID		*Atomicity, Consistency, Isolation, Durability*
PDF		*Portable Document Format*
CSV		*Comma-separated values*

1. INTRODUCCIÓN

Resumen: *En este capítulo se muestran los antecedentes de la información que es publicada por el sector salud, además de la problemática que representa concentrar estos formatos al ser procesados y analizados para ofrecer métricas sobre la interacción de afecciones y medicamentos, que aporten información simple de leer, en una interfaz web al usuario final. Se listan además los objetivos generales y específicos que incluyen cómo la información es procesada, analizada y presentada.*

1.1. Antecedentes

El sector salud ofrece abiertamente conjuntos de datos que son generados por diversas instituciones públicas de salud en la República Mexicana. La dirección general de información en salud (DGIS), el IMSS y el INEGI generan estos conjuntos de datos en una variedad de formatos como CSV, XLS y DOC, por mencionar algunos. Esto implica un esfuerzo en generar y almacenar dicha información, para después ser publicada, sin duda una tarea que requiere de recursos humanos y computacionales. Adicionalmente concentrar, procesar y analizar está información para proporcionar detalles sobre la misma es una tarea sumamente compleja para una persona con conocimientos nulos o básicos de cómputo.

1.2. Justificación

Debido a la diversidad de formatos en que la información es publicada, el volumen y el deseo de concentrar y transformar la información en conocimiento, se decidió crear un prototipo de herramienta de software capaz de procesar la información desde las diversas fuentes de datos y en sus distintos formatos, concentrarla, analizarla y así ofrecer métricas significativas que puedan ser de apoyo a los usuarios que toman de decisiones dentro del sector salud y para la población en general.

Este prototipo será desarrollado utilizando únicamente herramientas de código abierto orientadas al manejo de *big data*. Estas herramientas ofrecen diversas funcionalidades para hacer frente a esta problemática, facilitan el manejo de volúmenes grandes de datos con o sin estructura definida, además de estructuras dinámicas.

1.3. Problema

Los datos publicados son encontrados en archivos de Excel, texto, documentos PDF, en otras palabras, la estructura de estos archivos es diversa, en algunos de ellos hay estructuras definidas y en otros no hay una estructura definida.

El tamaño de almacenamiento requerido también representa un problema, la velocidad en la que el volumen de los datos crece, requiere de una infraestructura que ofrezca la continua transformación e integración de volúmenes grandes y con estructura dinámica.

Posteriormente se requiere de una infraestructura robusta para procesar toda esta cantidad de información, una arquitectura de sistema distribuido (o bien en la nube como Google Cloud) que permita el procesamiento y análisis eficaz de toda esta información.

Una vez que está información ha sido transformada y concentrada, los análisis deben ser realizados mientras los datos son procesados, de este modo se obtienen análisis en tiempo real y no es necesario

esperar a procesar los datos, almacenarlos y posteriormente realizar los análisis requeridos lo cual representa un consumo de tiempo bastante considerable.

Ya realizado el procesamiento y análisis, se requiere de una base de datos que permita analizar las interconexiones de los datos. Debido a la estructura dinámica de la información, esta base de datos no relacional debe soportar el manejo de estructuras dinámicas y con un modelo de datos basado en grafos, además de ofrecer un lenguaje de consultas que permita el análisis de interconexión entre los datos.

Finalmente presentar los análisis realizados al usuario final, mediante una interfaz web. El problema se puede abreviar entonces de la siguiente forma:

1. Transformar, almacenar y concentrar los datos con diversas estructuras en una arquitectura de sistema distribuido.
2. Procesar y analizar eficazmente la información con herramientas de *big data* que permitan el manejo de la información, con un modelo basado en grafos.
3. Almacenar el resultado del procesamiento y análisis en una base de datos no relacional, que permita el manejo de estructuras dinámicas y con un modelo de datos basado en grafos.
4. Mostrar los resultados al usuario final, en una interfaz web.

1.4. Objetivos

1.4.1. Objetivo general

Desarrollar un prototipo funcional capaz de recolectar información de atenciones hospitalarias publicadas en la página web de la DGIS, analizar la interacción entra afecciones y medicamentos mediante un modelo de grafo que permita identificar si existen relaciones entre ellas y su grado de incidencia.

1.4.2. Objetivos específicos

1.4.2.1 Extraer la información publicada en la en la página web de la DGIS, cargar los archivos en HDFS

1.4.2.2 Utilizar la herramienta TALEND para transformar los archivos en los distintos formatos y concentrar los archivos transformados en HDFS.

1.4.2.3 Procesar y analizar la interacción de afecciones y medicamentos, utilizando *Apache Hadoop, Spark SQL* y *GraphX*

1.4.2.4 Realizar consultas utilizando Neo4j con el lenguaje de consultas *cypher* mediante servicios *rest*.

1.4.2.5 Desarrollar una aplicación web que muestre los resultados de la interacción de afecciones y medicamentos.

1.5. Novedad científica, tecnológica o aportación

Actualmente realizar estudios sobre la interacción de los recursos dentro del sector salud y además de problemas de salud con medicamentos suministrados, no es una tarea muy simple. Se requieren de diversos conocimientos computacionales para realizar dichos análisis. Este prototipo mediante el uso de diversos componentes de código abierto tiene como principal aportación, ofrecer una representación de la información publicada en un formato simple y entendible para el usuario final. Esto, sin necesidad de contar con conocimientos computacionales especializados y mucho menos de minería de datos.

Este proyecto además integra una diversidad de herramientas de código abierto enfocadas al manejo masivo de información que hoy en día es referenciado como *big data*. Una de estas herramientas, es de reciente aparición y sigue en continuo crecimiento llamada *Apache Spark*, esta herramienta encapsula una serie de algoritmos de ML y de procesamiento de grafos, que funcionan tanto en una sola computadora como en una red de ellas y que es la infraestructura distribuida que este proyecto requiere.

2. ESTADO DEL ARTE O DE LA TÉCNICA

Resumen: *En este capítulo se presenta un resumen de los trabajos relacionados con el análisis de la información en temas de salud pública.*

2.1. Herramientas de análisis de datos en el sector salud

2.1.1 Observatorio de Salud Colombia

Entidad pública de salud [1] que actualmente tiene diversos proyectos de investigación en la toma de decisiones del sector salud, mediante el análisis de encuestas realizadas a la población. Entre la información publicada se puede encontrar las técnicas de análisis que utilizan para analizar los datos resultado de dichas encuestas. La red de conocimiento [2], es una técnica que se basa en el análisis de grafos y que explota cada una de las características que un grafo ofrece (Centralidad del nodo, Agrupamientos, cercanías, etc.). Hace uso de un algoritmo propuesto por Ulrik Brandes [3] para encontrar la centralidad del nodo con el programa Gephi [4] como se muestra en la Figura 1.

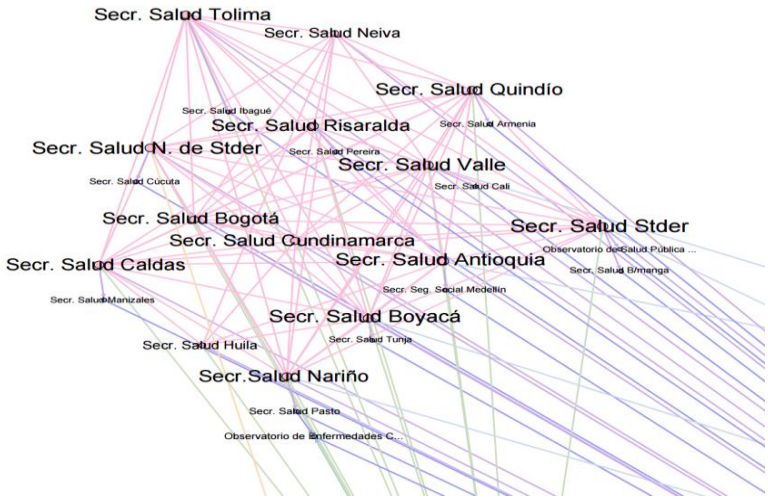


Figura 1. Red de conocimiento en Gephi [2]

El observatorio de salud ofrece una aplicación web para obtener datos estadísticos sobre las causas de mortalidad de la población. La Figura 2 muestra el formulario de los datos de entrada que la aplicación necesita para generar el reporte de datos. Se selecciona el país (por el momento el único valor disponible es Colombia), posteriormente se selecciona de una amplia lista de padecimientos, la causa de mortalidad y el método de cálculo.

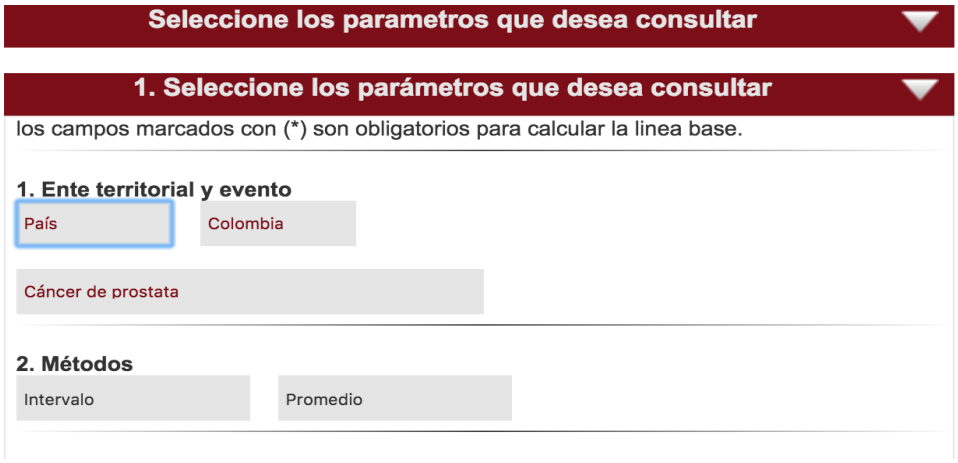


Figura 2. Mortalidad Evitable [1]

En la Figura 3 se muestra el formulario para ingresar los años de intervalo, mismos que ayudarán a generar un cálculo de la línea base.

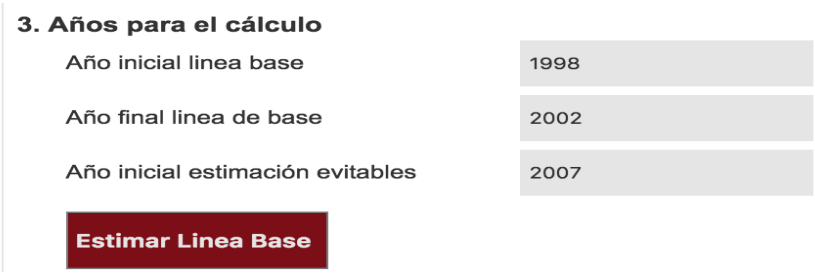


Figura 3. Mortalidad Evitable, selección de intervalo [1]

Una vez llenado el formulario, se genera el reporte como se muestra en la Figura 4, el reporte se muestra en formato de tabla y contiene la información distribuida por género y edad principalmente.

Grupo de edad	Población		Muertes observadas		Muertes esperadas		Muertes en exceso y déficit	
	Hombre	Mujer	Hombre	Mujer	Hombre	Mujer	Hombre	Mujer
menores 1 año	435.572	414.509	0	0	0	0	0	0
1 a 4 años	1.760.856	1.686.318	0	0	0	0	0	0
10 a 14 años	2.300.857	2.192.788	0	0	2	0	2	0
15 a 19 años	2.196.745	2.089.187	0	0	2	0	2	0
20 a 24 años	1.928.584	1.926.267	0	0	2	0	2	0
25 a 29 años	1.697.414	1.767.838	3	0	2	0	-1	0
30 a 34 años	1.495.687	1.582.244	2	0	2	0	0	0
35 a 39 años	1.424.205	1.536.754	3	0	3	0	0	0
40 a 44 años	1.385.527	1.502.566	4	0	7	0	3	0
45 a 49 años	1.204.613	1.314.696	10	0	18	0	8	0
5 a 9 años	2.250.467	2.162.092	0	0	0	0	0	0
50 a 54 años	973.726	1.063.049	30	0	33	0	3	0
55 a 59 años	773.759	840.313	62	0	68	0	6	0
60 a 64 años	588.669	642.939	126	0	149	0	23	0
65 a 69 años	456.293	516.646	249	0	259	0	10	0
70 a 74 años	350.326	415.995	398	0	399	0	1	0
75 a 79 años	230.171	280.409	481	0	410	0	-71	0
mayores de 80	229.600	309.248	1.030	0	1.004	0	-26	0
Muertes en exceso o déficit por Cáncer de prostata en Colombia, 2007			**2.398**	**2.398**	**2.360**	**2.360**	**-38**	**-38**

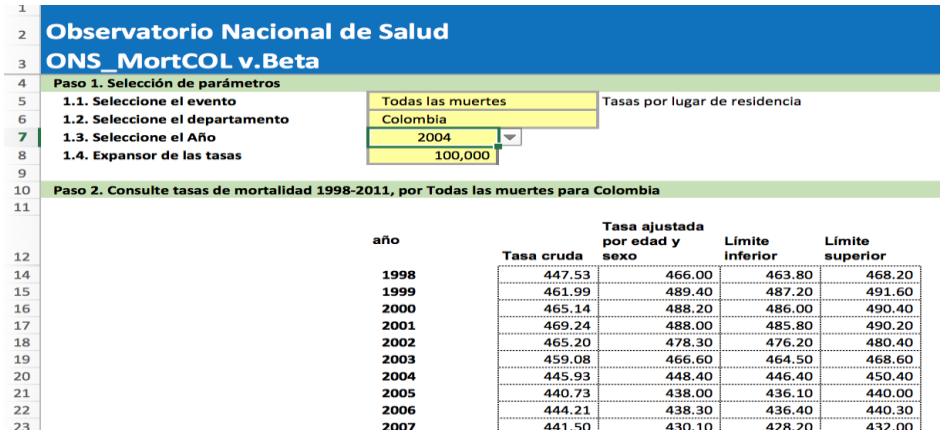
Figura 4. Reporte de los campos seleccionados

Al realizar la ejecución del reporte, se observa el campo de “Estimar Línea Base” (Figura 5).

Grupo de edad	Hombre	Mujer
menores 1 año	0	0
1 a 4 años	0	0
10 a 14 años	8,99264421150292E-07	0
15 a 19 años	1,04046398519131E-06	0
20 a 24 años	1,30834240508193E-06	0
25 a 29 años	1,52558904649898E-06	0
30 a 34 años	1,50498908624286E-06	0
35 a 39 años	2,02106386382184E-06	0
40 a 44 años	5,55504576027488E-06	0
45 a 49 años	1,46362403938838E-05	0
5 a 9 años	0	0
50 a 54 años	3,44593963745865E-05	0
55 a 59 años	9,12272662390023E-05	0
60 a 64 años	0,000251887940976303	0
65 a 69 años	0,00056431200937368	0
70 a 74 años	0,00111246688175015	0
75 a 79 años	0,00177966989576817	0
mayores de 80	0,00440648687072098	0

Figura 5. Detalles de la línea Base

Adicional a esta aplicación, se puede descargar un archivo de Excel indicando que se trata de una versión Beta (Figura 6). Se hace uso de macros para generar el reporte y los datos vienen ya cargados en las hojas de Excel.



Observatorio Nacional de Salud
ONS_MortCOL v.Beta

Paso 1. Selección de parámetros

1.1. Seleccione el evento	Todas las muertes	Tasas por lugar de residencia
1.2. Seleccione el departamento	Colombia	
1.3. Seleccione el Año	2004	
1.4. Expansor de las tasas	100,000	

Paso 2. Consulte tasas de mortalidad 1998-2011, por Todas las muertes para Colombia

año	Tasa cruda	Tasa ajustada por edad y sexo	Límite inferior	Límite superior
1998	447.53	466.00	463.80	468.20
1999	461.99	489.40	487.20	491.60
2000	465.14	488.20	486.00	490.40
2001	469.24	488.00	485.80	490.20
2002	465.20	478.30	476.20	480.40
2003	459.08	466.60	464.50	468.60
2004	445.93	448.40	446.40	450.40
2005	440.73	438.00	436.10	440.00
2006	444.21	438.30	436.40	440.30
2007	441.50	430.10	428.20	432.00

Figura 6. Versión beta, reporte de mortalidad

2.1.2 Observatorio Mexicano de Enfermedades no transmisibles

La Universidad Autónoma de Nuevo León nos ofrece OMENT [5] una herramienta en línea llamada *Sistema de Indicadores*, con una interfaz sencilla y que ofrece indicadores limitados por estado, la Figura 7 muestra la interfaz de selección de parámetros.

Figura 7. OMENT Formulario de indicadores

Ofrece un total de 4 pilares de estrategia (Atención médica entre ellos) y 3 diferentes componentes de análisis. Una vez creado el reporte como se muestra en la Figura 8, se muestran campos con información breve y muy general.

Figura 8. OMENT, Resultado de Indicadores

En la Figura 9 se muestra un reporte de indicadores concentrado por estado, el cual muestra los indicadores de referencia críticos en problemas de salud que actualmente la población padece. Esté reporte es un archivo PDF y por el momento los parámetros para ser ejecutado no son configurables.

Figura 9. OMENT Indicadores por Estado

El alcance de este trabajo con respecto a la información utilizada se centra en la información publicada por el sector salud en México. Este proyecto hace uso de una librería que facilita el manejo, análisis masivo y distribuido (en una red de computadoras) de grafos llamada *GraphX*, que *Gephi* no ofrece, además, el volumen de datos que *Gephi* puede procesar, es muy reducido.

La información que ofrece el observatorio mexicano de enfermedades no transmisibles, a pesar de ser de suma importancia, esta herramienta presenta algunas limitaciones. El reporte de indicadores ofrece parámetros muy limitados y se pueden obtener únicamente documentos PDF con datos e indicadores comparativos entre las distintas entidades federativas, como se muestra en la Figura 9.

El prototipo ofrece la parametrización del reporte con diversas afecciones y medicamentos reportados por el sector salud. Incluye información de interacción entre afecciones y medicamentos, lo cual muestra, por un lado, cómo los medicamentos son prescritos para las distintas afecciones y, por otro lado, con análisis de grafos, si hay alguna afección que sea la causante de otra o bien su grado de incidencia.

Se hace uso de una base de datos NoSQL, Neo4j, que permite el almacenamiento de la interacción entre afecciones y medicamentos, con un modelo de datos basado en grafos y como tal, obtener información de los nodos cercanos o bien del camino más corto entre ciertas afecciones, con su lenguaje de consultas *cypher*. Finalmente, este proyecto hace uso de herramientas de *big data*, lo que permite la integración continua de información publicada por el sector salud.

3. MARCO TEÓRICO

Resumen: *En este capítulo se presentan las bases teóricas y conceptuales sobre big data, herramientas de código abierto y mecanismos de almacenamiento usados en entornos big data, y cómo convergen para el análisis de interacción entre afecciones y medicamentos.*

Esta sección establece los conceptos que sirven de sustento para el desarrollo de este proyecto. La base fundamental de este proyecto es el manejo de datos masivos de información por lo tanto se enumera el conjunto de tecnologías utilizadas y su aportación en entornos *big data*, mismas que establecen el contexto en el que este proyecto se desarrolla.

3.1. BIG DATA

La información que es generada día a día por los sistemas de información, es de volúmenes grandes y de estructuras diversas, esto genera nuevos retos para procesar esta información; *Big data* por su nombre en inglés, en general es un término que hace referencia a un conjunto de datos masivos que no puede ser adquirido, administrado y procesado de una manera eficiente por herramientas convencionales de software y hardware [6]. Si bien hay diversas concepciones de la definición de *big data*, hay coincidencia en sus características relacionadas a los datos referidas como el modelo de las 4V's.

El **Volumen** de los datos ha crecido de manera exponencial ya que los sistemas generan demasiada información en muy poco tiempo, por ejemplo, la plataforma de videos *YouTube* registra aproximadamente 400 horas de video cargadas por los usuarios cada minuto, considerando la diversidad de resoluciones que la plataforma ofrece, el volumen de estos datos puede alcanzar la escala de *TeraBytes* o bien *PetaBytes* [7].

Variedad indica las diversas estructuras que contienen los diferentes tipos de datos, de las que se enumeran como semi-estructurada y no estructurada. Los datos relacionados con estas estructuras son en su mayoría imágenes, videos, archivos de audio, archivos de texto, páginas web o correos electrónicos y la información estructurada que es muy bien procesada con bases de datos relacionales. Además de los tipos de datos, la variedad refiere también al conjunto de dispositivos o herramientas que generan información como sensores, teléfonos móviles, redes sociales etc.

Velocidad comúnmente señala la velocidad con la que los datos son generados, transportados, almacenados o bien extraídos, sin embargo, en entornos de *big data* está velocidad está asociada a la capacidad de las tecnologías relacionadas en su manejo, en realizar análisis de la información en tiempo real, lo cual ofrece un panorama distinto, analizar la información mientras está siendo transportada, en lugar de analizarla cuando ya fue almacenada. La figura 10 muestra las dimensiones de estas características y como el volumen y la velocidad de los datos han ido cambiando desde sistemas tradicionales a entornos de *big data*. Dado que existen diversas concepciones en la definición de *big data*, hay más características que pueden ser encontradas.

Valor se refiere al conocimiento que puede ser extraído de estos, la toma de decisiones en los distintos sectores empresariales se fundamenta en el conocimiento que los datos nos pueden ofrecer como predicciones de ventas para el próximo semestre, predicciones climatológicas, tendencias en redes sociales etc.

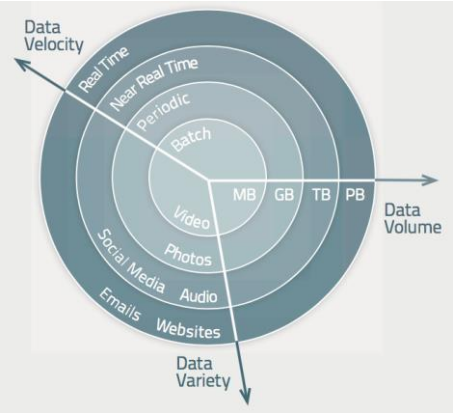


Figura 10. Las 3Vs de big data [8]

Veracidad se asocia a la confiabilidad de los datos, esta característica puede asociarse al proceso de análisis de los datos, si bien la minería de datos requiere de una consistencia de datos con cierto grado de confiabilidad para generar modelos predictivos o realizar análisis precisos, esta confiabilidad no es requerida para la estructura de los datos, su volumen o procesamiento, pero si requerida para obtener el valor.

Variabilidad está relacionada con el flujo de datos, con los datos en movimiento, los datos pueden cambiar en algún instante de tiempo, la estructura es diferente, algún campo de información nuevo etc. Está variabilidad es asociada también a la obtención del valor de los datos y no exclusivamente ya que comparte contexto con la variedad de los datos respecto a la estructura.

El *NIST (National Institute for Standards and Technology* por sus siglas en inglés), ofrece una definición que engloba el contexto de *big data* incluyendo a grandes rasgos la arquitectura que requiere.

"*Big data consiste en amplios conjuntos de datos con las características de volumen, velocidad, variedad y/o variabilidad que requieren de una arquitectura escalable para el eficiente almacenamiento, manipulación y análisis."* [9].

En secciones posteriores de este documento, se establece el tipo de arquitectura para el almacenamiento, procesamiento y análisis en *big data*, que este proyecto requiere. La Figura 11 muestra finalmente las 4V's asociadas a *big data* y las que este proyecto toma como fundamento, es importante considerar el valor del conocimiento que puede ser obtenido de los datos para la toma de decisiones.

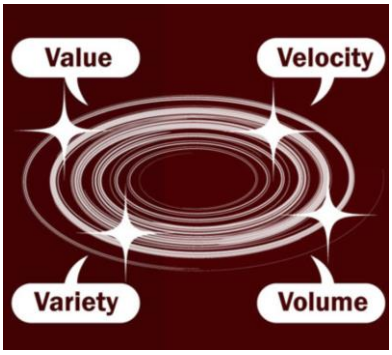


Figura 11. Las 4Vs de Big data [10]

3.2. Procesamiento de *big data*

Este prototipo, requiere de una arquitectura distribuida que permita el almacenamiento, procesamiento y análisis de la información del sector salud. En esta sección, se detallan las bases de las tecnologías que son usadas en la realización de este prototipo y que están orientadas a entornos *big data*.

3.2.1. MapReduce

El *framework MapReduce* ha sido introducido como un modelo de programación poderoso, permite el desarrollo de aplicaciones fácilmente escalables y paralelas que pueden procesar grandes cantidades de datos, en un gran clúster de computadoras comunes y corrientes. Esto sin preocuparse por la complejidad de la distribución de los datos entre cada nodo del clúster [7] . En este modelo de programación, la base principal, es un conjunto de tuplas llave-valor de entrada y de salida.

Este modelo funciona con la ejecución de dos funciones: Map y Reduce. La función *Map* toma la entrada y el framework realiza agrupaciones a partir de dicha llave, generando un conjunto de tuplas intermedias que son pasadas a la función de *Reduce*. Esta función toma el conjunto de tuplas y por cada llave recibida, realiza los agrupamientos y genera las tuplas de salida [7]. Un ejemplo de este modelo es el conteo de palabras en distintos documentos de texto. La Figura 12, muestra un diagrama de la ejecución de ambas funciones.

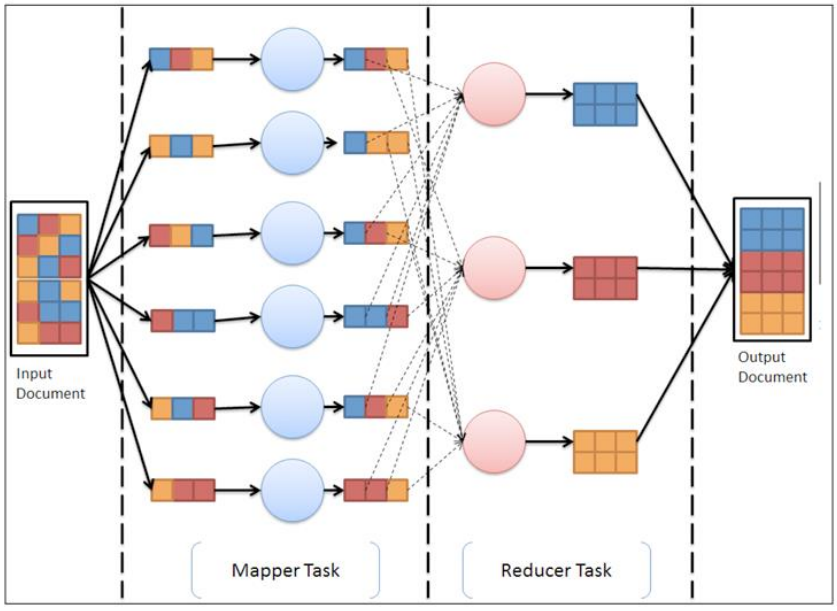


Figura 12. Ejecución de funciones *Map* y *Reduce* [11]

3.2.2. Apache Hadoop

Hadoop es un proyecto de código libre, que implementa el framework de MapReduce. A Nivel de implementación, las invocaciones Map, en un programa MapReduce, son distribuidas a cada nodo del clúster. Los datos son particionados automáticamente y con esto cada bloque de datos puede ser procesado en paralelo por distintos nodos [12].

Las invocaciones a la función de *Reduce*, son también distribuidas a cada nodo del clúster y en esta etapa, se particionan las tuplas intermedias y se les aplica la función *Reduce*. Ambas funciones, *Map* y *Reduce*, son definidas por el usuario.

Una de las limitaciones de este framework, requiere, que las tuplas de salida de cada llamada a las funciones *Map* y *Reduce*, sean materializadas en el *Hadoop Distributed File System* HDFS, antes de ser enviada a la siguiente etapa [7]. Lo cual afecta significativamente, el desempeño en el procesamiento de grandes cantidades de datos.

3.2.3. Hadoop Distributed File System

HDFS, es un sistema de archivos distribuidos, esto quiere decir que el almacenamiento es manejado en una red de computadoras, las cuales conforman el clúster de *Hadoop*. Este sistema de archivos es diseñado para almacenar una cantidad enorme de datos, lo cual encaja perfectamente en entornos *big data*. Una particularidad de este diseño, debido a que la posibilidad de que un nodo en el clúster falle es muy alta, es la habilidad de *Hadoop* al recuperar o reconstruir el bloque de archivo perdido [13].

Los archivos en HDFS, son divididos por bloques (128MB por default) y cada bloque es almacenado de manera independiente y distribuido entre los diversos nodos del clúster a partir de un valor de réplica (tradicionalmente de tres), que incrementa la facilidad de recuperar un bloque perdido y que esto sea transparente al usuario.

Un clúster de *Hadoop*, para el funcionamiento de su HDFS, tiene dos tipos de nodos con base en el patrón maestro-esclavo, *namenode* y *datanode*. El *namenode*, se encarga de mantener el árbol del sistema de archivos junto con todo el *metadata* de archivos y directorios, además guarda información de los *datanodes*, a los cuales se ha distribuido cada uno de los bloques de un determinado archivo.

Por otro lado, los *datanodes*, almacenan y recuperan los bloques de archivos, cuando el *namenode* lo requiera. Por lo tanto, sin un *namenode*, el sistema de archivos no podrá ser utilizado y por este motivo, se requiere tener un nodo de respaldo, a partir del cual se puede reconstruir el sistema de archivos. Este nodo de respaldo es conocido como *secondary namenode*. La Figura 13, muestra el diagrama de interacción de los nodos en un HDFS y como los bloques son replicados.

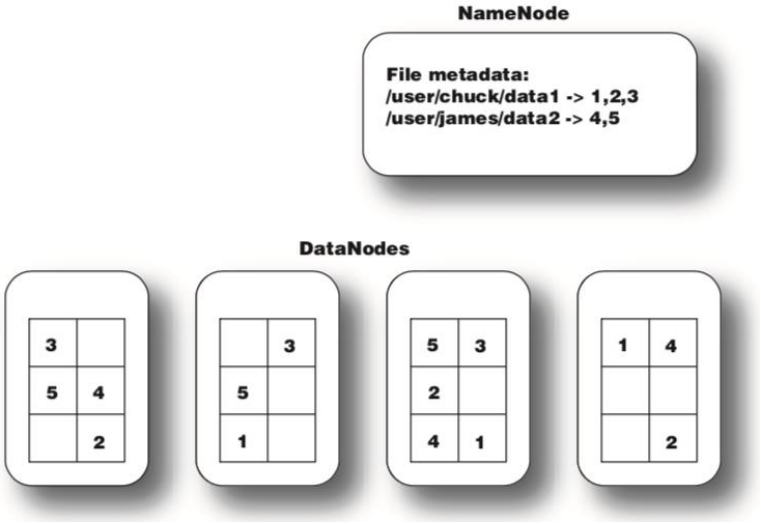


Figura 13. Replica de bloques de archivos en HDFS [14]

3.2.4. Apache Spark

Spark ha sido introducido como un motor de procesamiento de propósito general en entornos de *big data*, basado en el flujo del framework *MapReduce*, la principal abstracción del modelo de programación en *spark*, es nombrada *Resilient Distributed Datasets* (RDD). A diferencia de *Hadoop* que persiste los datos en HDFS, spark realiza la distribución de los datos en memoria distribuida, lo que permite, rapidez de acceso a cada uno de los conjuntos de datos particionados y distribuidos a cada nodo del clúster [15]. La Figura 14, muestra la diferencia de ambos modelos, cada paso, representa alguna operación *MapReduce* aplicada a los datos o tuplas intermedias.

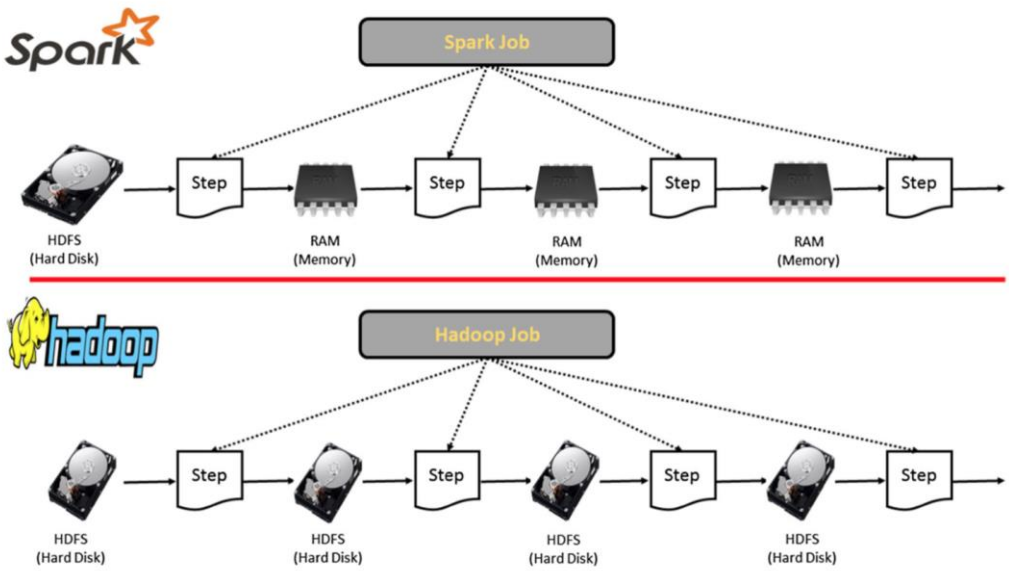


Figura 14. Spark vs Hadoop [16]

En principio, tener un RDD en memoria, permite al programador aplicar el paradigma de programación funcional, donde se indica que se quiere hacer, en comparación de cómo debe hacerse. Cada función de *Map* o *Reduce* en un RDD, resulta en un nuevo RDD, con las transformaciones previamente aplicadas.

La ventaja que ofrece este modelo, un RDD no será materializado en memoria, hasta que se haya aplicado alguna transformación en él, por ejemplo, una operación de conteo *count* o de conteo por valor *countByValue* [17]. Spark puede ser integrado, con otras herramientas para el manejo de *big data*, particularmente, programas escritos para spark, pueden ser ejecutados en un clúster de Hadoop y leer datos cuya fuente sea una HDFS. La Figura 15 muestra los componentes que conforman spark. Este prototipo, se centra en el uso de *spark sql* y *GraphX*.

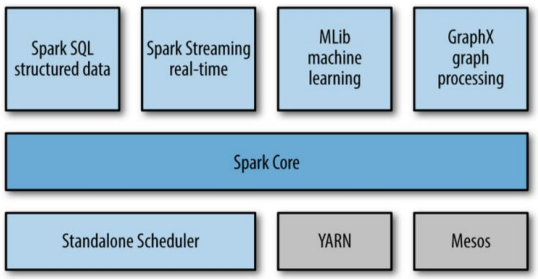


Figura 15. Components de spark [18]

Spark SQL, es una librería para trabajar con datos estructurados, permite el acceso a los datos mediante consultas SQL, lo cual permite al desarrollador, integrar consultas SQL, que dan como resultado un RDD y operaciones de transformación sobre estos.

3.2.5. Spark GraphX

Es una librería para el procesamiento distribuido de grafos. Extiende de la abstracción del RDD, para implementar *Resilient Distributed Graph* (RDG). Un RDG representa un grafo dirigido, con una estructura de adyacencia, que es mapeado como registros en una vista tabular de vértices y arista [19]. La Figura 16, muestra la representación de un RDG en GraphX.

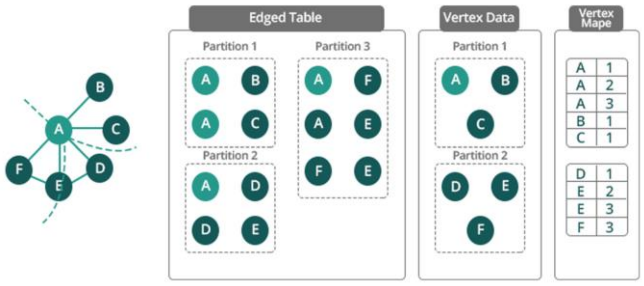


Figura 16. Representación de un grafo en GraphX [20]

Dado que un RDG extiende de la abstracción de un RDD, la transformación de un RDG da como resultado un nuevo RDG. Adicionalmente esta librería, permite la composición de los grafos a partir de datos no estructurados y de datos tabulares, permitiendo que el mismo conjunto físico de datos, pueda ser visto como un grafo y como una colección de datos (Figura 17).

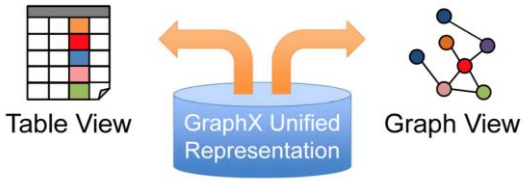


Figura 17. Composición de grafos en GraphX [21]

GraphX implementa un modelo de datos basado en grafos nombrado *labeled property graph*, en la siguiente sección, se muestran las bases de este modelo y la relación que tiene además con la base de datos usada en este prototipo *Neo4j*.

3.2.6. Modelo de datos basado en grafos

Un modelo de datos basado en grafos puede ser definido como aquel en el que la estructura de los datos tanto para el esquema y las instancias son modelados como un grafo o como una generalización de estos. Las operaciones de manipulación tienen como base la transformación del grafo o bien por operaciones primitivas basadas en el recorrido del grafo donde puede obtenerse información como los caminos más cortos, vértices vecinos, sub-grafos, conectividad del grafo y estadísticas (diámetro, centralidad etc.) [22].

Por lo tanto, un modelo de tipo *labeled property graph*, refiere a un modelo de datos basado en grafos, donde el usuario define propiedades y etiquetas, a cada nodo y a cada arista [23]. Por ejemplo, una propiedad en un nodo de tipo afección, puede ser la temporada en donde la afección se presenta con mayor frecuencia. Si fuese un resfriado, la temporada sería invierno. Por otro lado, una etiqueta, es un identificador en el nodo o en una relación, que permite agrupar a los nodos o relaciones, e indicar cuál es el rol que tienen en el conjunto de datos [24].

La Figura 18 muestra un grafo con propiedades y etiquetas, este modelo de datos permite una fácil integración entre *GraphX* y la base de datos usada en este prototipo Neo4j, que hace uso del mismo modelo de datos.

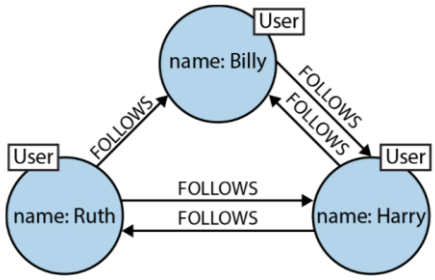


Figura 18. Ejemplo de un *Labeled property graph* [25]

3.2.7. Neo4J

Es una base de datos clasificada como NoSQL, que ofrece nativamente almacenamiento y procesamiento de los datos, con un modelo de grafo [26]. A diferencia de otras bases de datos NoSQL, Neo4j ofrece soporte completo transaccional, considerando las propiedades *Atomicity*, *Consistency*, *Isolation*, *Durability* (ACID).

Atomicity refiere a la capacidad de ejecutar un bloque de operaciones en unidad, si alguna de las operaciones falla, el resto no tendrá efecto alguno. *Consistency* en operaciones de escritura, el resto de los

clientes que ejecuten operaciones de lectura, obtendrán el resultado de la última escritura realizada. *Isolation* operaciones transaccionales están completamente aisladas, lo que significa, que una transacción de escritura no afectará alguna otra transacción en operaciones de lectura. *Durability* los datos escritos en la base de datos, estarán disponibles después de un reinicio o de algún fallo en el servidor. La Figura 19, sumariza cada una de estas propiedades.

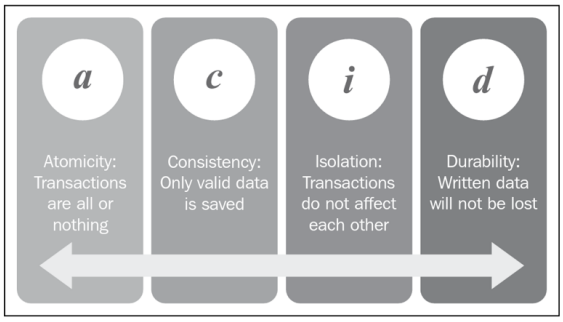


Figura 19. Propiedades ACID [27]

Neo4j es una base de datos desarrollada con *java*, por lo tanto, ofrece una API, desarrollada en el mismo lenguaje, para la manipulación de nuestros datos. Esto sin duda, requiere de habilidades y de conocimiento en ese lenguaje y adicionalmente la escritura de consultas complejas hará que el código en java para realizarlas sea aún más complejo. En la siguiente sección, se muestra el lenguaje de consultas usado por Neo4j *cypher* y que facilita el acceso y la manipulación de nuestros datos.

3.2.8. Cypher

Es un lenguaje de consulta declarativo inspirado en SQL para describir patrones en los grafos y permite definir qué es lo que deseamos consultar, insertar, actualizar o borrar de nuestro grafo, sin la necesidad de definir cómo hacerlo. La Figura 20 muestra la relación entre nodos y cómo se escribiría la consulta con cypher.

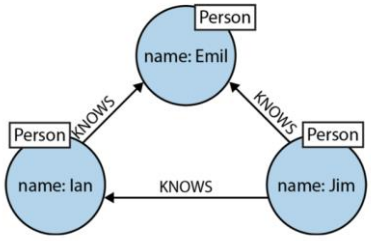


Figura 20. Relación de nodos en Neo4j [28]

Cypher ha sido diseñado para ser fácilmente leído y entendido, la consulta fue escrita de la misma forma en la que se dibujó el diagrama. De aquí se observa que los nodos están entre paréntesis y las relaciones son indicadas con corchetes. Como se mencionó anteriormente, el modelo de datos en Neo4j, pueden tener propiedades y etiquetas. En el ejemplo mostrado, Person es la etiqueta que le da contexto a cada nodo, name es una propiedad del nodo y finalmente, la relación esta etiquetada también.

Esto permite realizar consultas, indicando la etiqueta que se desea filtrar. El siguiente ejemplo considera que existe, otra etiqueta para la relación entre personas del tipo WORK_WITH, para obtener a todas las personas que trabajan con Jim, la siguiente consulta escrita en cypher, regresaría tal información.

```
match(p:Person {name:'Jim'})<-[:WORK_WITH]-(coworker:Person) return coworker
```

4. DESARROLLO METODOLÓGICO

Resumen: *En este capítulo se presenta en detalle el desarrollo metodológico que incluye la obtención de los conjuntos de datos relacionados con las afecciones y medicamentos, los requerimientos necesarios para extraer, transformar y cargar la información procesada en un entorno distribuido como spark, para posteriormente hacer el análisis de interacción entre afecciones y medicamentos utilizando GraphX. Se detalla el procedimiento de carga de datos en Neo4j y la interfaz web para el usuario final. Finalmente, se presenta el diagrama de arquitectura seleccionada para la solución.*

4.1. Metodología propuesta

La información publicada por el sector salud de manera abierta es diversa. Este prototipo se centra en el análisis de la interacción de afecciones y medicamentos que es publicada en la sección de urgencias. Para realizar el análisis correspondiente de dicha interacción, se propone que los datos sean manejados como grafos y que sean almacenados como tal.

El procedimiento general es el siguiente:

1. Descargar el conjunto de datos en formato *CSV*.
2. Extraer los campos de interés (afecciones y medicamentos), y almacenar, con el mismo formato, en el sistema de archivos distribuidos de *Hadoop*.
3. Utilizando *Spark SQL* transformar los archivos creando tablas, para generar el grafo con *GraphX*.
4. Aplicando los algoritmos *pageRank* y *connected components*, analizar la interacción de afecciones y medicamentos con *GraphX*. Almacenar los resultados del análisis en tablas de *spark sql*.
5. Cargar la interacción de afecciones y medicamentos, del paso anterior, a la base de datos *Neo4j*
6. Hacer uso de los servicios *REST* expuestos por *Neo4j* para la ejecución de consultas con el lenguaje *cypher* y mostrar los resultados en una aplicación web para el usuario final.

A continuación, se detalla cada uno de los pasos.

4.1.1. Descargar el conjunto de datos en formato CSV

La dirección general de información en salud, publica en el siguiente enlace la información a utilizar en este proyecto.

Fuente: http://www.dgis.salud.gob.mx/contenidos/basesdedatos/Datos_Abiertos_gobmx.html

En la sección de urgencias están listados los registros de urgencias del periodo que comprende los años 2008 al 2015. Los archivos que son procesados para la extracción de los campos son: urgencias, afecciones y medicamentos.

El proceso de *ETL* es realizado con la herramienta *TALEND*, los campos que son extraídos son: Clave de la entidad, año, afección y medicamento. La Figura 21 muestra el flujo creado en *TALEND* para la extracción de estos campos.

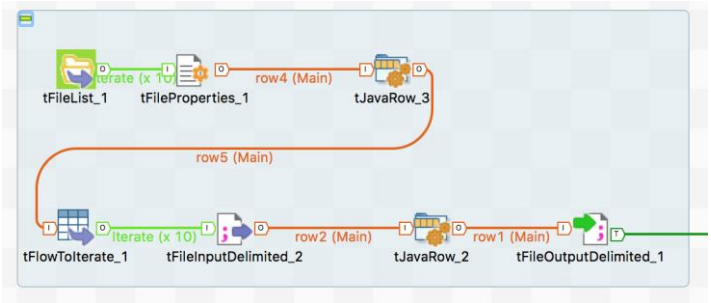


Figura 21. Flujo de extracción en *TALEND*

Los archivos resultantes son cargados en *Hadoop*. Este proyecto aplica una arquitectura de solución basada en la nube ofrecida por *google*, en particular, el componente que implementa este sistema de archivos es *google buckets.* La Figura 22 muestra los archivos ya cargados en *google buckets.*

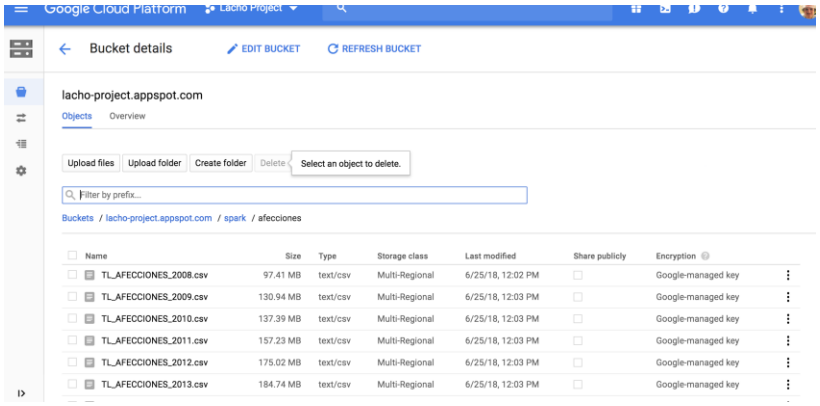


Figura 22. Archivos cargados en *google buckets*

El siguiente paso es extraer los campos de interés año, clave de afección y clave de medicamento, de los archivos previamente procesados y almacenar el resultado con el mismo formato de CSV, en el sistema de archivos distribuidos de Hadoop.

Debido a la facilidad que ofrece *spark sql* de manejar archivos de texto delimitados como si estos fueran tablas en una base de datos relacional, las siguientes tablas fueron creadas y cargadas con datos de los archivos previamente transformados y cargados a *google buckets*.

- Afecciones
- Medicamentos

Una vez cargados los datos, conectándonos a la terminal interactiva del nodo maestro de nuestro clúster *Hadoop-Spark*, ofrecido por *google cloud* bajo el nombre de *google data proc. L*a Figura 23 muestra el código usado para la creación de la tabla de urgencias y medicamentos.

```
CREATE EXTERNAL TABLE default.afecciones(
id int,
afeccion string,
anio_reporte int)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' STORED AS TextFile
LOCATION 'gs://lacho-project.appspot.com/spark/afecciones/TL_AFECCIONES*.csv'

CREATE EXTERNAL TABLE default.medicamentos(
id int,
medicamento int,
anio_reporte int)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' STORED AS TextFile
LOCATION 'gs://lacho-project.appspot.com/spark/medicamentos/TL_MEDICAMENTOS*.csv'
```

Figura 23. Creación de tablas en *spark sql*

Con este par de tablas, se simplifica el manejo de los archivos de texto delimitados por comas y ahora se puede acceder a la información realizando consultas tradicionales utilizando SQL. Buscando simplificar aún más el procesamiento de estos datos, se realiza la creación de una tercera tabla que cruce la información de afecciones y medicamentos. Contendrá las columnas id y de anio_reporte.

Se crea esta última tabla como estrategia de diseño y así evitar operaciones excesivas de map-reduce al crear los RDDs de la tabla de afecciones y de la tabla de medicamentos. Con esto, además, se reducen tiempos de ejecución en los análisis de pageRank y de connected components que son realizados con GraphX. El diagrama final de tablas puede verse en la Figura 24.

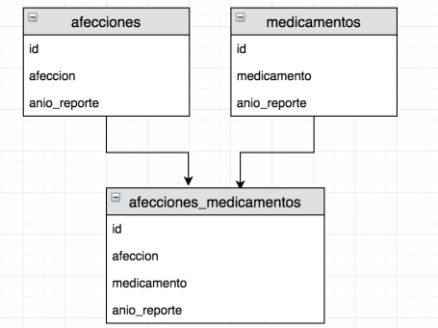


Figura 24. Modelo de tablas en *spark sql*

4.1.2. Utilizando Spark SQL transformar los archivos creando tablas, para generar el grafo con GraphX.

El análisis de la interacción entre afecciones y medicamentos se lleva a cabo con la librería de *spark GraphX*. El par de algoritmos que se requiere ejecutar hace uso de una estructura de datos basada en grafos, la Figura 25 muestra el esquema del grafo que va a ser procesado.

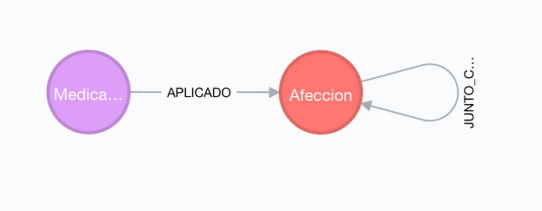


Figura 25. Esquema de grafo afecciones-medicamentos

En los datos publicados por el sector salud, se muestra que un paciente puede ser tratado por diversas afecciones, un ejemplo de esto puede ser un paciente que acude con una infección en las vías respiratorias

y además presenta dolor en garganta y oídos. El modelo de grafo presentado no limita el número de afecciones que pueden encadenarse entre sí.

Como argumentos de entrada para crear el grafo, *GraphX* recibe una lista de nodos (afecciones y medicamentos) y una lista de aristas. Uno de los requerimientos de esta librería, es que los identificadores del nodo sean numéricos. Nodos y aristas son creados mediante consultas SQL a la tercera tabla creada previamente.

Una vez creados los RDDs de nodos y aristas, la siguiente línea de código escrita en *scala*, muestra la creación del grafo.

```
val grafo= Graph (nodosAfeccionesMedicamentos, aristas)
```

El algoritmo *pageRank* funciona inicializando cada nodo con un valor de 1/N, donde N es el número total de nodos en el grafo. Posteriormente se itera cada nodo y se envía al nodo adyacente el valor de 1/M, donde M es el número total de aristas de salida del nodo actual. Con esta información enviada al nodo adyacente, se suma y se obtiene un nuevo valor de *pageRank*. Se repite el proceso hasta que los valores de *pageRank* no han cambiado significativamente, por ejemplo, no más de 0.001. Para determinar este cambio, se recibe un parámetro de entrada que representa el valor de tolerancia. Este valor, es usado para determinar cuando el algoritmo debe detenerse. Este algoritmo genera los siguientes resultados, (considerando una pequeña muestra) mostrados en la Tabla 1.

ID	Rank
4882	589.33
2712	319.41

Tabla 1. Resultados de *pageRank*

Los valores 4882 y 2712, corresponden a las afecciones de hipertensión y diabetes respectivamente. Estos nodos fueron parte de los valores más altos reportados por el algoritmo. El valor de *rank* nos indica que, para ese nodo, la posibilidad de que exista un camino en el grafo iniciando desde algún nodo aleatorio, es alta.

El algoritmo *connected components*, funciona implementando una búsqueda de anchura en el grafo, se itera cada nodo y sus nodos adyacentes. Conforme se itera cada nodo y sus vecinos, se marcan como visitados y se crea un valor de partición que será asignado a cada par de vértices conectados en toda la búsqueda de anchura. El valor de partición se irá incrementando cuando no se haya visitado algún nodo en las iteraciones anteriores. Este algoritmo no recibe parámetros de entrada y la salida generada es mostrada en la Tabla 2.

ID	partición
4882	1
2712	1

Tabla 2. Resultados *connected components*

El id regresado refiere también a la afección, como en el algoritmo anterior. El valor de partición indica el grupo de pertenencia del nodo. Con este algoritmo se pueden identificar comunidades de afecciones.

La ejecución del algoritmo *pageRank* con la variable previamente definida, se ejecuta de la siguiente manera: *grafo.pageRank(0.0001)* y la ejecución del algoritmo *connected components grafo.connectedComponents.*

Una vez terminada la ejecución de cada algoritmo, los resultados son regresados en el mismo RDD que fue utilizado para la ejecución. Es necesario señalar que los análisis realizados con *GraphX* son por año (2008 al 2015), por lo tanto, los resultados de cada ejecución serán guardados en una tabla para ser consultados posteriormente con *spark sql.* El nuevo modelo de tablas puede apreciarse en la Figura 26.

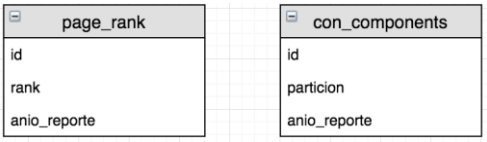


Figura 26. Modelo de tablas *pageRank* y *connected components*

4.1.3. Cargar la interacción de afecciones y medicamentos, del paso anterior, a la base de datos Neo4j

Neo4j cubre con el requerimiento de almacenar la interacción de afecciones y medicamentos, con un esquema de grafo. La carga es realizada mediante archivos CSV y usando la herramienta de carga de esta base de datos. La Figura 27 muestra el código utilizado para cargar la interacción de afecciones y medicamentos del año 2015.

```
USING PERIODIC COMMIT 100
LOAD CSV FROM 'file:///pad_afec_2015.csv' as line with split(line[0],";") as afecciones
UNWIND RANGE(0,size(afecciones)-2) as idx
WITH afecciones[idx] as af_1,afecciones[idx+1] as af_2
match(a:Afeccion {idaf:af_1})
match(b:Afeccion {idaf:af_2})
MERGE(a)-[:JUNTO_CON {anio:2015}]->(b)
```

Figura 27. Carga de afecciones a Neo4j

El lenguaje de consultas *cypher*, ofrece flexibilidad y diversos procedimientos que facilitan la carga de los datos. Durante la carga, cada línea del archivo de entrada puede ser transformada y el resultado de esa transformación usada para encontrar información ya existente en la base de datos.

El archivo anterior ha sido exportado desde *spark*, solo contiene una columna que muestra la relación de afecciones que son presentadas en conjunto por algún paciente. Como ejemplo, el primer registro contiene las siguientes claves de afecciones: L282;M549;J039, que representan las afecciones otros prurigos, dorsalgia y amigdalitis. Lo que quiere decir, que estas afecciones fueran presentadas en conjunto por un paciente y con esto, se crea un grafo, tal como se muestra en la Figura 28.

El proceso de carga trabaja de la siguiente manera. La columna es separada por el carácter ‘;’ y con esto un arreglo de cadenas es creado. La función *UNWIND* genera un iterador numérico que es usado para acceder a cada elemento del arreglo. Las funciones *MATCH* encuentran las afecciones en la base de datos (previamente cargadas) para posteriormente crean una relación entre ellas. La Figura 28 muestra en la interfaz web de *Neo4j*, cómo estas relaciones fueron creadas.

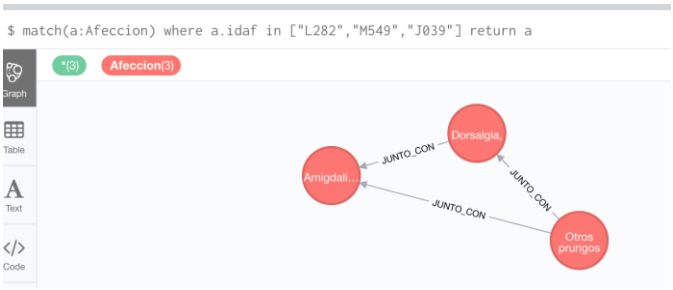


Figura 28. Grafo creado en Neo4j

4.1.4. Hacer uso de los servicios REST expuestos por Neo4j para la ejecución de consultas con el lenguaje cypher y mostrar los resultados en una aplicación web para el usuario final.

Finalmente, se propone una interfaz web para el usuario final. Este proyecto se ha centrado en el análisis de las interacciones entre afecciones y medicamentos en los distintos periodos, así que uno de los componentes que debe contener, es la selección de afecciones y medicamentos para consultar dicha interacción. La Figura 29 muestra la pantalla principal, se pueden seleccionar distintas afecciones y distintos medicamentos.

Figura 29. Interfaz web para usuario final

Esta interfaz web es construida con *Java* y desplegada en el servidor de aplicaciones *apache tomcat*. Para acceder a los datos almacenados en *Neo4j*, se hace uso del siguiente servicio *rest* desplegado en el mismo servidor donde reside nuestra base de datos: http://host_name:7474/db/data/cypher. El código en java de la Figura 30, muestra una consulta escrita en *cypher* que regresa los medicamentos que han sido aplicados a las diversas afecciones.

```
public List<Medicamento> getMedicamentos(){

    String medicamentosQuery=" match(m:Medicamento)-[:APLICADO]->(b:Afeccion) return  distinct m order by m.medicamento";
    String valores=getCatalogo(medicamentosQuery);
    ObjectMapper mapper= new ObjectMapper();
    TypeFactory typeFactory= mapper.getTypeFactory();
    List<Medicamento> result=null;

    try{
        result= mapper.readValue(valores,typeFactory.constructCollectionType(List.class,Medicamento.class));

    } catch (Exception e) {
        e.printStackTrace();
    }


    return result;

}
```

Figura 30. Consulta en *cypher* vía servicio rest

Para ejecutar correctamente la llamada al servicio *rest*, los parámetros deben ser proporcionados en formato *json* { query: “match(m:Medicamento)-[:APLICADO]->(b:Afeccion) return distinct m order by m.medicamento”}.

En la sección siguiente, se muestra el diagrama de la arquitectura utilizada

4.2. Arquitectura propuesta

Procesar una cantidad grande de información y que esta sea representada en un grafo para analizar la interacción entre afecciones y medicamentos, nos ha llevado a buscar una solución de arquitectura en la nube con *google cloud*. La facilidad para crear un clúster *Hadoop-Spark* y tenerlo listo para la ejecución de procesos bajo demanda, ha sido uno de los principales motivos para elegir este servicio. La Figura 31 muestra el diagrama de los componentes que conforman nuestra arquitectura de solución.

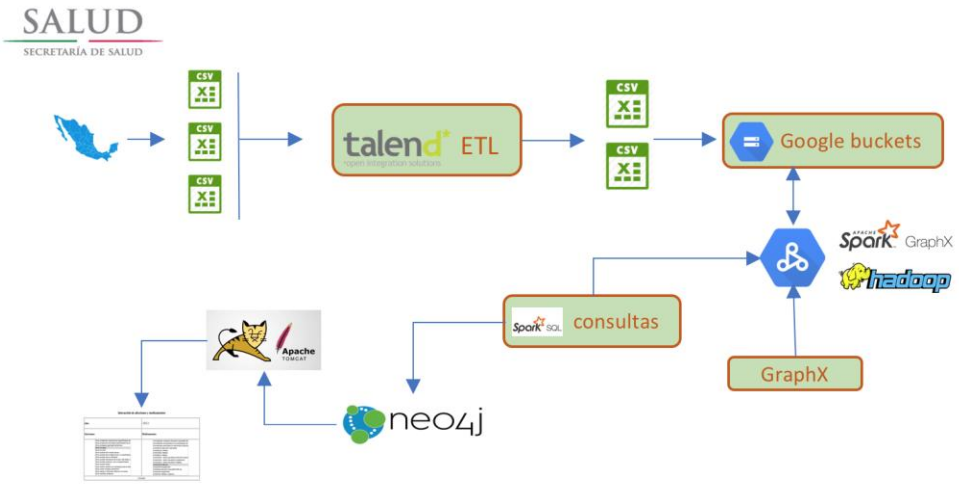


Figura 31. Arquitectura de solución con *google cloud*

Google dataproc es el servicio que ofrece el conjunto de componentes para crear nuestro clúster *Hadoop-Spark.* La configuración mínima sugerida por *google* es suficiente para la ejecución de análisis con *GraphX* y para la creación de tablas con *spark sql*, un nodo maestro y dos nodos esclavos. Una vez finalizado el procesamiento de los datos, se requiere un nodo para la ejecución de *Neo4j* y para el acceso de la interfaz web. *Compute engine* es el servicio mediante el cual se puede configurar este nodo e instalar los recursos necesarios.

En este último nodo a diferencia del clúster que ofrece todo un entorno ya configurado e interconectado con *google buckets* que implementa HDFS, los recursos que deben instalarse son la base de datos *Neo4j* y el servidor de aplicaciones *apache tomcat*. La Figura 32 muestra los servicios en ejecución de ambos recursos.

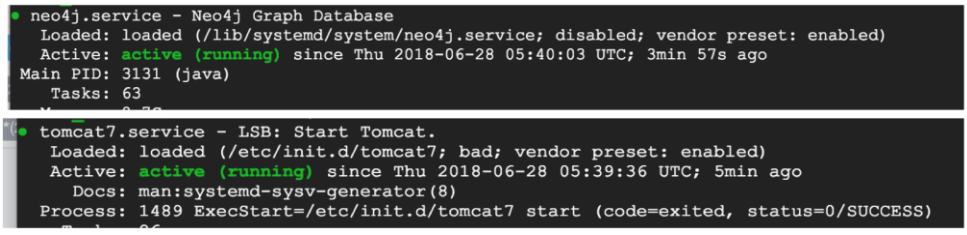


Figura 32. Recursos en ejecución *compute engine*

Por seguridad cualquier instancia creada por *compute engine*, expone únicamente el puerto 80 para acceso público y de cualquier lugar. *Neo4j* hace uso del puerto 7474 y 7686, *apache tomcat* el puerto 8080, estos

puertos deben ser expuestos en las reglas del *firewall*. La Figura 33 muestra la configuración que nos permitirá acceder a ambos recursos.

Applicable firewall rules | Applicable routes

Name	Type	Description	Targets	Filters	Protocols / ports	Action	Priority
default-allow-http	Ingress		http-server	IP ranges: 0.0.0.0/0	tcp:80, tcp:8080	Allow	1000
neo	Ingress	Conexion a neo4j	Apply to all	IP ranges: 0.0.0.0/0	tcp:7474	Allow	1000
neo-bolt	Ingress		Apply to all	IP ranges: 0.0.0.0/0	tcp:7687	Allow	1000

Figura 33. Reglas de *firewall*

5. RESULTADOS Y DISCUSIÓN

Resumen: *En este capítulo se presentan los resultados obtenidos del desarrollo de este trabajo y una discusión sobre la interacción entre afecciones y medicamentos reportadas por el sector salud.*

5.1. Resultados obtenidos con TALEND

Inicialmente, los archivos descargados del sector salud contienen columnas que por el momento no son utilizadas. El proceso de extracción y transformación obtiene únicamente las columnas de interés y las guarda con el delimitador '|'. La Tabla 3 y Tabla 4, muestran algunos ejemplos de los campos extraídos de los documentos de medicamentos y afecciones respectivamente.

ID de registro	Clave de medicamento	Nombre	Año de reporte
10004	1308	Metronidazol tableta	2008
25	3417	Diclofenaco capsula	2010

Tabla 3. Extracción de medicamentos

ID de registro	Clave de afección	Nombre	Año de reporte
7618820	S109	Traumatismo superficial del cuello	2015
7618821	J029	Faringitis aguda	2013

Tabla 4. Extracción de afecciones

5.2. Resultados obtenidos con *spark sql*

Ya creadas las tablas y cargadas con la información almacenada en *google buckets*, la Figura 34 muestra el total de registros que se tienen por año en medicamentos, afecciones y el cruce de estas dos tablas, esta tercera tabla es utilizada para reducir los tiempos de ejecución y reducir operaciones de *map-reduce* al realizar los análisis con *GraphX*.

```
scala> afecciones.show(false)
+------------+--------+
|anio_reporte|total   |
+------------+--------+
|2008        |5738347 |
|2009        |7694445 |
|2010        |8068128 |
|2011        |9226361 |
|2012        |10259818|
|2013        |10826906|
|2014        |11496898|
|2015        |11555824|
+------------+--------+

scala> medicamentos.show(false)
+------------+-------+
|anio_reporte|total  |
+------------+-------+
|2008        |2794097|
|2009        |3308526|
|2010        |3016413|
|2011        |2928795|
|2012        |3235217|
|2013        |3278359|
|2014        |3311461|
|2015        |3154965|
+------------+-------+

scala> afecciones_medicamentos.show(false)
+------------+-------+
|anio_reporte|total  |
+------------+-------+
|2008        |3171369|
|2009        |3849226|
|2010        |3511950|
|2011        |3414688|
|2012        |3802762|
|2013        |3870910|
|2014        |3977862|
|2015        |3733610|
+------------+-------+
```

Figura 34. Resultado afecciones y medicamentos

5.3. Resultados obtenidos con *GraphX*

El primer análisis realizado es *pageRank*, a continuación, se listan los primeros 10 registros en la Figura 35 con valores más altos reportados por el algoritmo. Los resultados obtenidos por el algoritmo de *connected components*, muestra las comunidades encontradas. Por el momento se presentan únicamente los tamaños de cada comunidad encontrada en la Figura 36.

rank	afeccion	nombre
589.338082411383	J068	Otras infecciones agudas de sitios múltiples de las vías respiratorias superiores
319.4146141360443	I10X	Hipertensión esencial (primaria)
163.06171570328195	E109	Diabetes mellitus insulinodependiente, sin mención de complicación
157.4706754553861	A020	Enteritis debida a Salmonella
132.5237976426179	K589	Síndrome del colon irritable sin diarrea
119.53588406945383	S069	Traumatismo intracraneal, no especificado
114.28793520057538	N23X	Cólico renal, no especificado
99.96895455158568	T009	Traumatismos superficiales múltiples, no especificados
95.39523466586736	Z915	Historia personal de lesión autoinfligida intencionalmente

Figura 35. Resultado de *pageRank*

```
res82: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[1410] a

scala> itercc.map(x=> x.toArray.size).collect
res83: Array[Int] = Array(7576)
```

Figura 36. Resultado de *connected components*

En el caso del algoritmo de *connected components*, no se encontraron demasiadas comunidades, solamente una, lo cual puede indicar que hay aristas entre todas las afecciones y medicamento, no hay afecciones ni medicamentos aislados. En la sección de discusión y con el apoyo de los resultados de *pageRank*, se contrastarán ambos resultados.

5.4. Resultados obtenidos con *Neo4j*

La Figura 37, muestra el resultado de las afecciones con mayor número de vecinos cercanos en el grafo.

```
$ match (a:Afeccion)-[:JUNTO_CON]->(b:Afeccion) return a.idaf,a.afeccion,count(b) as total order by total desc
```

Table | Text | Code

a.idaf	a.afeccion	total
"E119"	"Diabetes mellitus no insulinodependiente, sin mención de complicación"	688
"J029"	"Faringitis aguda, no especificada"	541
"D649"	"Anemia de tipo no especificado"	535
"R104"	"Otros dolores abdominales y los no especificados"	527
"A090"	"Otras gastroenteritis y colitis de origen infeccioso"	473

Figura 37. Afecciones y sus vecinos cercanos

Seleccionando la afección con identificador *E119* de este resultado, la Figura 38, muestra el resultado de los vecinos de los vecinos cercanos de la afección seleccionada. En otras palabras, se están contando los

nodos a un nivel de profundidad 2 a partir de la afección *E119*, es decir, cuando se presenta la afección E119, también se presentan las afecciones J029, K297, J068 y R509.

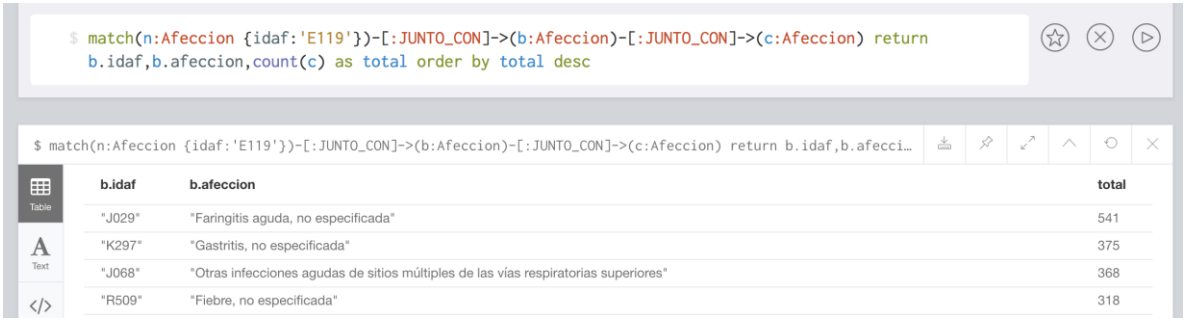


Figura 38. Vecinos a nivel de profundidad 2

Por otro lado, si deseamos conocer las afecciones en común entre una afección y sus vecinos, la Figura 39 muestra la consulta en cypher para obtener este resultado.

Figura 39. Afecciones en común con vecinos cercanos

Si deseamos obtener algunas de las afecciones relacionadas en más de un nivel de profundidad, se hace uso del algoritmo *shortestPath* proporcionado por *neo4j*, La Figura 40 muestra el resultado de algunas afecciones relacionadas entre la afección *S827* y *K055*.

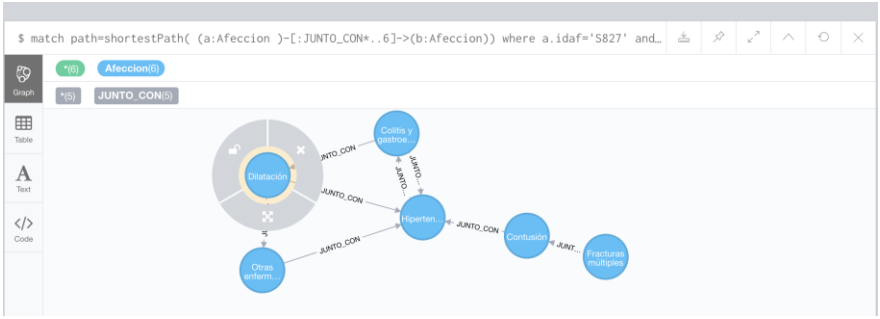


Figura 40. Camino más corto entre dos afecciones

Considerando dos afecciones del resultado de *pageRank*, una con valor alto y otra con un valor bajo, se realizan las siguientes consultas en *cypher*.

> match(a:Afeccion)<-[:JUNTO_CON*1..4]-(b:Afeccion) where a.idaf='J068' return distinct a.idaf, a.afeccion, count(b) as total
>
> match(a:Afeccion)<-[:JUNTO_CON*1..4]-(b:Afeccion) where a.idaf=T003' return distinct a.idaf, a.afeccion, count(b) as total

Estas consultas hacen un conteo de todos los nodos que están relacionados con estas afecciones, puede observarse que la consulta se hace a un nivel 4 de profundidad. La Figura 41 muestra el conteo de la afección con valor más bajo de *pageRank* y la Figura 42 muestra el valor más alto.

Figura 41. Afección con valor bajo de *pageRank*

Figura 42. Afección con valor alto de *pageRank*

Lo que quiere decir, que la afección T003, traumatismos superficiales, tiene una baja posibilidad de que, iniciando un camino desde un nodo aleatorio, este camino llegue o incluya, la afección T003. Por otro lado, la afección J068, otras infecciones agudas, tiene una alta posibilidad de aparecer como un nodo final o estar incluido.

En la siguiente sección, se realiza una discusión sobre el significado que tiene esta posibilidad en el análisis de la interacción de afecciones y medicamentos.

5.5. Discusión

Los resultados de *pageRank*, hasta este momento han mostrado ser de gran utilidad al analizar la interacción de las afecciones y medicamentos. Este algoritmo ofrece un panorama sobre la importancia que tiene un nodo en el grafo. Para nuestro caso de estudio, nos ofrece la posibilidad de identificar, si hay alguna afección que sea la causante de otra o bien su grado de incidencia.

Considerando los resultados de la Figura 42, haciendo uso de la interfaz web, la Tabla 5 muestra la selección de la afección y el medicamento. El medicamento ha sido seleccionado aleatoriamente, para mostrar que la funcionalidad de este prototipo pretende ser amigable y simple para un usuario que no es experto en el sector salud.

Afección	Medicamento
Otras infecciones agudas de sitios múltiples de las vías respiratorias …	Paracetamol Tableta

Tabla 5. Selección de afección y medicamento en interfaz web

Los resultados se dividen en tres secciones, la primera sección mostrada en la Figura 43, ofrece información sobre los medicamentos que han sido encontrados en los conjuntos publicados por el sector salud como prescritos a la afección seleccionada. La segunda sección mostrada en la Figura 44, muestra las afecciones a las cuales ha sido prescrito el medicamento o los medicamentos seleccionados en la interfaz web. Finalmente, la tercera sección mostrada en la Figura 45, muestra las afecciones en un nivel de profundidad uno (los vecinos cercanos), que están relacionadas con la afección o afecciones seleccionadas.

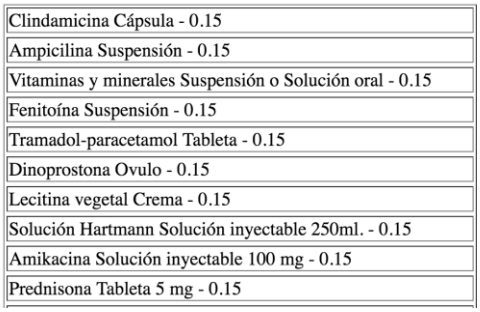


Figura 43. Otros medicamentos prescritos a la afección seleccionada

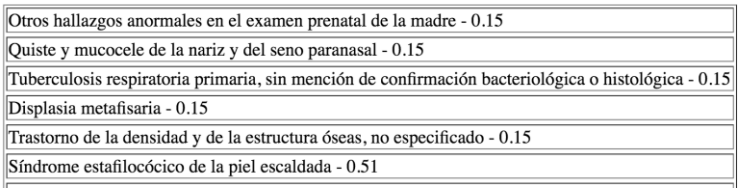


Figura 44. Otras afecciones relacionadas con el medicamento seleccionado

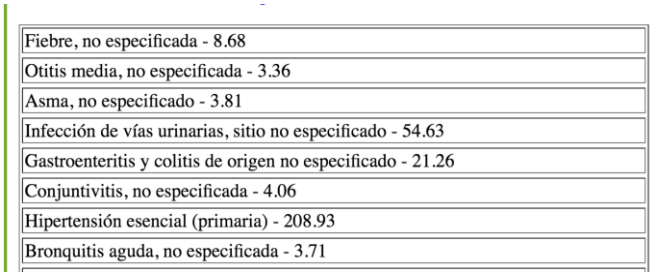


Figura 45. Otras afecciones relacionadas con la afección seleccionada

La información numérica que es mostrada en cada resultado representa el valor de *pageRank* obtenido previamente. El valor constante 0.15 mostrado, significa que la afección o el medicamento, no tiene ningún vecino cercano que incida en él. Todos los medicamentos tendrán este valor, debido al esquema mostrado en la Figura 25, el medicamento incide en las afecciones y no hay alguna entidad por el momento que incida en él. Hay algunas afecciones que muestran también este valor, la situación es similar, no hay alguna otra afección que incida en ellas.

El resultado de la Figura 45, nos muestra que otras afecciones han sido presentadas junto con la afección seleccionada, regularmente la fiebre es una afección presente junto con alguna infección, ya sea respiratoria o de otro tipo. Hay una afección y un valor que particularmente llama la atención debido a que es un valor muy alto en comparación a los demás, la hipertensión y su valor de *pageRank* de 208.93.

El sector salud, ha puesto especial atención en difundir la importancia de lo crónica que puede resultar la hipertensión y las afecciones que la acompañan, cómo la obesidad y la diabetes mellitus tipo 2. Al seleccionar la hipertensión esencial (primaria), en la interfaz web, la Tabla 6 muestra algunas de las afecciones relacionadas y que son de especial interés para la salud, con su respectivo valor de *pageRank.*

Afecciones
Diabetes mellitus no insulinodependiente, sin mención de complicación - 24.11
Otros trastornos de ansiedad especificados - 12.76
Migraña sin aura [migraña común] - 3.79
Infarto agudo de miocardio, sin otra especificación - 3.55
Otras enfermedades cerebrovasculares especificadas – 2.92

Tabla 6. Afecciones relacionadas con la hipertensión

Como se observa en los resultados, los valores de *pageRank* no son muy altos en comparación al valor de hipertensión, lo que quiere decir, que estos valores indican el grado de incidencia en el nodo, no solamente de sus vecinos cercanos, también del resto de nodos en el grafo. Esto indica que la hipertensión tiene un

grado muy alto de incidencia, lo que significa que es una afección que desencadena otras afecciones, como las que son mostradas en la Tabla 6.

Por otro lado, los resultados obtenidos por el algoritmo de *connected components*, y que se muestran en la Figura 36, no son satisfactorios, ya que la expectativa es identificar diversas comunidades de afecciones y medicamentos.

Los resultados muestran únicamente una comunidad, con un total de 7,576 nodos de afecciones y medicamentos, esto quiere decir que cada afección y cada medicamento en el grafo, al realizar la búsqueda por anchura, tiene al menos una arista con otro nodo.

En la sección de trabajo futuro, se recomienda una modificación al modelo del grafo mostrado en la Figura 25, podría permitir obtener resultados satisfactorios en la identificación de comunidades.

6. CONCLUSIONES

Resumen: *En este capítulo se presentan las conclusiones y trabajo futuro en relación, al análisis de interacción de afecciones y medicamentos.*

6.1. Conclusiones

Contar con una herramienta como TALEND, que permita la extracción, transformación y la carga de un conjunto de datos, ha sido parte fundamental para el desarrollo de este trabajo. A pesar de que inicialmente estos datos fueron descargados manualmente, se seleccionó esta herramienta para futuros desarrollos y mejoras, porque permite la carga y extracción de datos, desde un sistema de archivos distribuidos como lo es HDFS de *Hadoop.*

Ofrece interconectividad con diversos proveedores de servicio en la nube como *google* o *Amazon AWS*, que ofrecen servicios bajo demanda de clústeres *Hadoop-spark.* En caso de que el proveedor del servicio en la nube cambie, la modificación de los procesos de carga de TALEND, no requerirán de mucho esfuerzo.

El sistema de archivos distribuidos facilitó la interacción de *spark sql* y *GraphX*, para este proyecto, fue usado el servicio de *google buckets*. Si bien el costo aproximado de almacenamiento mensual por gigabyte es de $0.026 USD, este servicio ahorra el costo de infraestructura del clúster completo de *Hadoop-spark*, que incluye un nodo maestro y dos nodos esclavos, considerando por cada uno el costo de almacenamiento, procesadores y memoria RAM.

El modelo de grafo usado sirvió para obtener resultados satisfactorios en la ejecución del algoritmo *pageRank.* A pesar de que hubo resultados en la ejecución del algoritmo *connected componentes,* estos resultados no muestran una clara detección de comunidades entre afecciones y medicamentos, que, aunque estaban fuera del alcance de este trabajo, sería interesante analizar en otro proyecto.

La base de datos seleccionada *Neo4j*, ofrece una facilidad para la consulta de los datos con su lenguaje *cypher*, además de herramientas que facilitan la carga de datos a una base de datos nueva o existente. La consulta de datos mediante el servicio *rest*, permite la extracción de datos en consultas sencillas, debido a que el formato de respuesta es en *json. N*o se recomienda el uso del servicio *rest* cuando las consultas sean demasiado complejas, ya que esto agrega demasiada complejidad al procesar la respuesta.

La Interfaz web, interconectada con Neo4j mediante el servicio *rest*, muestra resultados satisfactorios en la interacción de afecciones y medicamentos con ayuda del resultado de *pageRank*. Esta interfaz, por el momento, no está aprovechando el potencial que ofrece Neo4j para la consulta del camino más corto entre dos afecciones o para la función inversa, obtener el camino más largo entre dos nodos, que permitiría obtener información sobre la interacción de afecciones, sin importar el nivel de profundidad. Está ultima funcionalidad, para el objeto de estudio de este proyecto, mostraría las diversas afecciones que son presentadas, por ejemplo, junto con la hipertensión y la diabetes.

Se cumple el objetivo 1.4.1 *Objetivo general*, analizar las interacciones entre afecciones y medicamentos, con un modelo de grafo. El algoritmo de *pageRank,* permitió identificar si hay afecciones causantes de otras, además de su grado de incidencia.

6.2. Trabajo Futuro

Este proyecto se centró en el análisis de interacción entre afecciones y medicamentos. El modelo de grafo utilizado mostró resultados notables en la ejecución de *pageRank*. Sin embargo, el algoritmo de *connected components*, puede ofrecer resultados variados si se modifica el modelo de la siguiente manera.

Actualmente, el medicamento es incidente a la afección y la afección es incidente a otras afecciones, por lo tanto, se podrían identificar comunidades de afecciones, si el medicamento no es utilizado en el algoritmo.

Otra posible modificación, si se desea mantener el modelo actual, sería incluir la información de la entidad, edad y genero del paciente atendido. Con esta información, se puede ampliar el modelo del grafo y realizar de nueva cuenta la ejecución de los algoritmos de *pageRank* y de *connectedComponents* y así, obtener información detallada por estado, por edad o por género. Esta mejora implica modificar el proceso de extracción y carga con TALEND, las tablas creadas en *spark sql*, la carga de datos a Neo4j y las consultas realizadas por la interfaz web.

Otro trabajo futuro que se recomienda sería incorporar otros algoritmos para el análisis de grafos. Neo4j ofrece la flexibilidad de agregar librerías de terceros para ejecutar ciertos análisis. Específicamente, la librería *apoc-procedures* entrega la funcionalidad para la detección de comunidades, *betweenness centrality* y *closeness centrality*, que pueden proporcionar más información sobre la interacción de afecciones y medicamentos.

Finalmente, se podría agregar funcionalidad a la interfaz web, que permita consultar las afecciones relacionadas en un nivel de profundidad seleccionado por el usuario. Lo que implica crear nuevas consultas y procesar los resultados que serán mostrados en la interfaz.

BIBLIOGRAFÍA

[1] Instituto Nacional de Salud Colombia, «Observatorio Nacional de Salud,» [En línea]. Available: http://www.ins.gov.co/. [Último acceso: Julio 2018].

[2] Instituto Nacional de Salud Colombia, «Redes de Conocimiento,» [En línea]. Available: http://simposiovirologia.ins.gov.co/lineas-de-accion/ons/Paginas/redes-de-conocimiento.aspx. [Último acceso: Julio 2018].

[3] U. Brandes, "Computer & Information Science," [Online]. Available: http://algo.uni-konstanz.de/members/brandes/.

[4] Gephi, «The Open Graph Viz Platform,» [En línea]. Available: https://gephi.org/. [Último acceso: Julio 2018].

[5] V. Sitalakshmi, Kiran, Fahd, K. Samuel y V. Ramanathan, «SQL Versus NoSQL Movement with Big Data Analytics,» *International Journal of Information Technology and Computer Science*, vol. 8, nº 12, pp. 59-66, 2016.

[6] K. Hyun-Woo, P. Jong Hyuk y J. Young-Sik, «Human-centric storage resource mechanism for big data on cloud service architecture,» *The Journal of Supercomputing*, vol. 72, nº 7, pp. 2437-2452, 2016.

[7] B. Fuad, E. Radwa, B. Omar, A. Abdulrahman, B. Ahmed y S. Sakr, «Big Data 2.0 Processing Systems: Taxonomy and Open Challenges,» *Journal of Grid Computing*, vol. 14, nº 3, pp. 379-405, 2016.

[8] P. CIO, «Big Data's 3vs [Figura]. En Big Data Big Promise. p. 7,» e.Republic, 2013.

[9] National Institute of Standards and Technology, [En línea]. Available: https://www.nist.gov/. [Último acceso: Julio 2018].

[10] C. Min, M. Shiwen y L. Yunhao, «The 4Vs feature of big data [Figura]. En Big Data a survey,» *Mobile Networks and Applications*, vol. 19, nº 2, pp. 171-209, 2014.

[11] K. Hrishikesh, MapReduce approach [Figura]. En Scaling Big Data with Hadoop and Solr Second Edition, p. 3, Packt publishing, 2015.

[12] L. Chuck, Hadoop in Action, Manning, 2011.

[13] T. White, Hadoop The Definitive Guide, O'Reilly, 2015.

[14] L. Chuck, NameNode/DataNode interaction in HDFS [Figura]. En Hadoop in Action, p. 23, Manning, 2011.

[15] K. Holden, K. Andy, W. Patrick y Z. Matei, Learning Spark, O'Reilly, 2015.

[16] B. Fuad, E. Radwa, B. Omar, A. Abdulrahman, B. Ahmed y S. Sakr, «Spark Framework vs Hadoop Framework [Figura]. En Big Data 2.0 Processing Systems: Taxonomy and Open Challenges, p. 383,» *Journal of Grid Computing*, vol. 14, nº 3, pp. 379-405, 2016.

[17] B. Marko y Z. Petar, Spark In Action, Manning, 2016.

[18] K. Holden, K. Andy, W. Patrick y Z. Matei, The Spark stack [Figura]. En Learning Spark, p. 3, O'Reilly, 2015.

[19] V. Aleksa y W. Nicki, Neo4j in Action, Manning, 2015.

[20] B. Fuad, E. Radwa, B. Omar, A. Abdulrahman, B. Ahmed y S. Sakr, «RDG Representation of Graphs in GraphX [Figura]. En Big Data 2.0 Processing Systems: Taxonomy and Open Challenges, p. 396,» *Journal of Grid Computing*, vol. 14, nº 3, pp. 379-405, 2016.

[21] B. Fuad, E. Radwa, B. Omar, A. Abdulrahman, B. Ahmed y S. Sakr, «Unified Representation of Graphs in GraphX [Figura]. En Big Data 2.0 Processing Systems: Taxonomy and Open Challenges, p. 397,» *Journal of Grid Computing*, vol. 14, nº 3, pp. 379-405, 2016.

[22] A. Renzo, «A Comparison of Current Graph Database Models,» de *ICDEW '12 Proceedings of the 2012 IEEE 28th International Conference on Data Engineering Workshops*, 2012.

[23] Z. Matei, R. X. W. Patrick, D. Tathagata y A. Michael, «Apache Spark: A Unified Engine for Big Data Processing,» *Communications of the ACM*, vol. 59, nº 11, pp. 56-65, 2016.

[24] I. Robinson, W. Jim y E. Emil, Graph Databases new opportunities for connected data, O'Reilly, 2015.

[25] I. Robinson, W. Jim y E. Emil, A small social graph [Figura]. En Graph Databases new opportunities for connected data, p. 2, O'Reilly, 2015.

[26] A. Goel, Neo4j cookbook, Packt Publishing, 2015.

[27] V. B. Rik, Acid-compliant database [Figura]. En Learning Neo4j, p. 45, Packt publishing, 2014.

[28] I. Robinson, W. Jim y E. Emil, A simple graph pattern, expressed using a diagram [Figura]. En Graph Databases new opportunities for connected data, O'Reilly, 2015.

[29] C. Min, M. Shiwen y L. Yunhao, «Big Data: A Survey,» *Mobile Networks and Applications*, vol. 19, nº 2, pp. 171-209, 2014.

[30] I. C. Alexandru, «Big Questions on Big Data,» *Revista de Cercetare si Interventie Sociala*, vol. 55, pp. 112-126, 2016.

[31] Fundación Carlos Slim, «Observatorio de la salud, Medir para Innovar,» [En línea]. Available: https://www.salud.carlosslim.org. [Último acceso: Julio 2018].

[32] J. E. Gonzalez, X. Reynold S., A. D. C. Daniel, F. Michael J. y S. Ion, «GraphX: Graph Processing in a Distributed Dataflow Framework,» de *11th USENIX Symposium on Operating Systems Design and Implementation*, 2014.

[33] K. Hrishikesh, Scaling Big Data with Hadoop and Solr Second Edition, Packt publishing, 2015.

[34] Instituto Mexicano del Seguro Social, «IMSS,» [En línea]. Available: http://datos.imss.gob.mx/. [Último acceso: Julio 2018].

[35] Instituto Nacional de Salud Colombia, «Aplicativo para el análisis sobre la mortalidad,» [En línea]. Available: http://aplicacionespruebas.ins.gov.co/MortalidadEvitable/. [Último acceso: Julio 2018].

[36] M. Michael, B. Mark, N. Cassie y G. Joy, «NoSQL Database Technologies,» *International Technology and Information Management*, vol. 24, nº 1, pp. 1-14, 2015.

[37] M. Michael y E. Robin, GraphX in Action, Manning, 2016.

[38] P. CIO, «Big Data Big Promise,» e.Republic, 2013.

[39] A. Rezo y C. Gutierrez, «Survey of Graph Database Models,» *ACM Computing Surveys*, vol. 40, nº 1, pp. 1-39, 2008.

[40] V. B. Rik, Learning Neo4j, Packt publishing, 2014.

[41] Universidad Autónoma de Nuevo León, «Observatorio Mexicano de Enfermedades No Transmisibles,» [En línea]. Available: http://oment.uanl.mx/. [Último acceso: Julio 2018].

[42] Wolfram, «Wolfram Language & System,» [En línea]. Available: https://reference.wolfram.com/language/ref/Cubics.html. [Último acceso: Julio 2018].

[43] P. Zikopoulos, C. Eaton, T. Deutsch, D. Deroos y L. George, Understanding Big Data, McGrawHill, 2012.