



UNIVERSIDAD CENTRAL DE VENEZUELA
FACULTAD DE CIENCIAS
ESCUELA COMPUTACIÓN
CENTRO DE COMPUTACIÓN DISTRIBUIDA Y PARALELA

MANEJO DE GRAFOS DE GRAN ESCALA UTILIZANDO APACHE SPARK Y GRAPHX

TRABAJO ESPECIAL DE GRADO PRESENTADO ANTE LA ILUSTRE

UNIVERSIDAD CENTRAL DE VENEZUELA POR EL

BR. ERIC GABRIEL BELLET LOCKER.

C.I. 24.463.483.

TUTORES: JESÚS LARES Y JOSÉ SOSA.

CARACAS, OCTUBRE 2016.

Acta

UNIVERSIDAD CENTRAL DE VENEZUELA
FACULTAD DE CIENCIAS
ESCUELA COMPUTACIÓN

ACTA DEL VEREDICTO

Quienes suscriben, Miembros del Jurado designado por el Consejo de la Escuela de Computación para examinar el Trabajo Especial de Grado, presentado por el bachiller Eric Gabriel Bellet Locker C.I.: 24.463.483, con el título “Manejo de grafos de gran escala utilizando Apache Spark y GraphX”, a los fines de cumplir con el requisito legal para optar al título de Licenciado en Computación, dejan constancia de lo siguiente:

Leído el trabajo por cada uno de los Miembros del Jurado, se fijó el día <día> de Octubre de 2016, a las <hora>, para que sus autores lo defendieran en forma pública, en el aula, lo cual estos realizaron mediante una exposición oral de su contenido, y luego respondieron satisfactoriamente a las preguntas que les fueron formuladas por el Jurado, todo ello conforme a lo dispuesto en la Ley de Universidades y demás normativas vigentes de la Universidad Central de Venezuela. Finalizada la defensa pública del Trabajo Especial de Grado, el jurado decidió aprobarlo.

En fe de lo cual se levanta la presente acta, en Caracas el <día> de Octubre de 2016, dejándose también constancia de que actuó como Coordinador del Jurado el Profesor Tutor Jesús Lares.

Prof. Jesús Lares
Tutor

Prof. José R. Sosa
Co-tutor

Prof. Hector Navarro
Jurado

Prof.
Jurado

Agradecimientos

A mi padre Alain, madre Gabrielle y hermana Isabelle gracias por su apoyo incondicional.

A mis tutores Jesús Lares y José Sosa, gracias por su apoyo durante todo el proyecto y por siempre estar al tanto de este gran trabajo. Un especial agradecimiento al profesor Héctor Navarro por facilitarme un entorno de trabajo y estar atento a mis avances.

A todos mis amigos de la universidad Andrés Álvarez, Carlos Zapata, Karen Moncada, Miguel Del Duca, Rafael Piña, Leonardo Santella, Deyban Pérez, Fernando Crema, Juan Sanchez, David Pérez, Pedro Valdivieso, Sebastian Ziegler e Israel Rodríguez.

A la Universidad Central de Venezuela, a la Escuela de Computación y a todos y cada uno de los profesores que contribuyeron en mi formación académica.

Gracias a todos aquellos que durante esta carrera fueron de gran apoyo y me ayudaron a crecer como profesional y persona.

Resumen

Título: Manejo de grafos de gran escala utilizando Apache Spark y GraphX.

Autor: Eric Bellet.

Tutor: Prof. Jesús Lares y José Sosa.

Las organizaciones comúnmente utilizan herramientas básicas o tradicionales para el análisis de datos, sin embargo estas no son capaces de almacenar grandes volúmenes, no permiten representar de diversas maneras la gran variedad de datos y no los procesan de forma rápida y eficiente, por lo tanto existen muchos datos que no pueden ser manipulados correctamente o simplemente no son tratados. Gracias a los avances tecnológicos actualmente existen herramientas que permiten el manejo de grandes volúmenes de datos (Big Data), por ejemplo Apache Hadoop. Big Data se aplica para toda aquella información que no puede ser procesada o analizada utilizando procesos o herramientas tradicionales. Existen muchas formas de modelar los conjuntos de datos, una de estas es denominada grafo, que permite representar redes de la vida cotidiana.

En consecuencia, el objetivo de este trabajo especial de grado, consiste en estructurar una red que represente un problema de grandes volúmenes de datos, como un grafo de gran escala, para posteriormente mediante múltiples operaciones almacenarlo, realizar cálculos, implementar algoritmos y técnicas de visualización para obtener información sencilla y compleja sobre la red. Las herramientas utilizadas fueron HDFS, Apache Spark, GraphX, Scala y Gephi. HDFS es un sistema de archivo distribuido, que permite que los datos no se guarden en una única máquina sino que sea capaz de distribuir la información en distintos dispositivos. Spark es una plataforma de computación de código abierto para análisis y procesos avanzados, que tiene muchas ventajas sobre Hadoop MapReduce. GraphX es un API para grafos que permite el manejo de estos de forma paralela. Scala es un lenguaje de programación multiparadigma que combina propiedades de lenguajes funcionales con orientados a objetos. Gephi es una herramienta para visualizar redes. Los conceptos y aplicaciones desarrollados en este trabajo especial de grafos pueden ser utilizados en cualquier tipo de red, sin embargo debido a que es evidente la gran cantidad de datos que manejan las redes sociales, se utilizó una red social para las pruebas de las herramientas mencionadas anteriormente.

Palabras clave: Big Data, Grafo, Apache Hadoop, Apache Spark, GraphX, Scala, Gephi.

Índice general

Acta	II
Agradecimientos	III
Resumen	IV
Índice de figuras	X
Índice de tablas	XIII
1. Introducción	1
2. Problema de investigación	2
2.1. Planteamiento del problema	2
2.2. Justificación	3
2.2.1. ¿Por qué es un problema?	3
2.2.2. ¿Para quién es un problema?	3
2.2.3. ¿Desde cuándo es un problema?	3
2.3. Objetivos de la investigación	3
2.3.1. General	3
2.3.2. Específicos	3
2.4. Antecedentes	4
2.5. Alcance	4
2.6. Consideraciones	5
2.6.1. Metodología	5
2.6.2. Tecnologías a utilizar	5
3. Marco teórico	6
3.1. Dato	6
3.2. Información	6
3.3. Conocimiento	6
3.4. Minería de datos (Data Mining)	7
3.4.1. Minería de grafos (Graph Mining)	8
3.5. Aprendizaje automático (Machine Learning)	8
3.6. Inteligencia artificial (Artificial Intelligence)	9
3.7. Inteligencia de negocios (Business Intelligence)	9

3.8. Ciencia de datos	9
3.8.1. Big data	9
3.8.2. Campos en los cuales es utilizado	11
3.9. Base de datos	12
3.9.1. Base de datos relacionales	13
3.9.1.1. Ventajas de las base de datos relacionales	13
3.9.1.2. Desventajas de las base de datos relacionales	13
3.9.2. Base de datos no relacionales	13
3.9.2.1. Ventajas de las base de datos no relacionales	14
3.9.2.2. Desventajas de las base de datos no relacionales	14
3.9.2.3. Tipos de base de datos no relacionales	15
3.9.2.3.1. Orientada a columnas	15
3.9.2.3.2. Orientada a clave/valor	16
3.9.2.3.3. Orientada a documentos	16
3.9.2.3.4. Orientada a grafo	16
3.10. Almacén de datos (Data Warehouse)	17
3.11. Lenguajes de programación	17
3.11.1. R	18
3.11.1.1. R Studio	18
3.11.1.2. Igraph	19
3.11.2. Java	19
3.11.3. Scala	19
3.11.4. Python	20
3.12. Apache Hadoop	21
3.12.1. Common	22
3.12.2. Hadoop Distributed File System (HDFS)	23
3.12.2.1. Características	23
3.12.2.2. Arquitectura	23
3.12.3. Yet Another Resource Negotiator (YARN)	25
3.12.3.1. Características	25
3.12.3.2. Arquitectura	25
3.12.4. MapReduce	26
3.12.4.1. JobTracker	27
3.12.4.2. TaskTracker	27
3.12.4.3. Map	28
3.12.4.4. Reduce	28
3.12.5. Ecosistema Hadoop	28
3.12.5.1. Administración de los datos	29
3.12.5.2. Acceso a los datos	29
3.12.5.2.1. Apache Accumulo	29
3.12.5.2.2. Apache Hbase	29
3.12.5.2.3. Apache Hive	31
3.12.5.2.4. Apache Storm	32
3.12.5.2.5. Apache Mahout	32
3.12.5.3. Integración	34

3.12.5.3.1. Apache Falcon	34
3.12.5.3.2. Apache Flume	34
3.12.5.3.3. Apache Sqoop	34
3.12.5.4. Operaciones	34
3.12.5.4.1. Apache Ambari	35
3.12.5.4.2. Apache Zookeeper	35
3.12.5.4.3. Apache Oozie	35
3.12.5.5. Seguridad	35
3.12.5.5.1. Apache Knox	36
3.12.5.5.2. Apache Ranger	36
3.12.6. Apache Spark	36
3.12.6.1. Spark SQL	37
3.12.6.2. MLlib	37
3.12.6.3. Spark streaming	38
3.12.6.4. Grafo Acíclico Dirigido (Directed Acyclic Graph (DAG))	38
3.12.6.5. Conjuntos Distribuidos Resilientes (Resilient Distributed Dataset (RDD))	39
3.12.6.6. Ventajas de Spark	40
3.12.7. Distribuciones Hadoop	41
3.12.7.1. Cloudera	41
3.12.7.2. Hortonworks	42
3.12.7.3. MapR	42
3.12.7.4. Cloudera vs Hortonworks	43
3.13. Grafo	44
3.13.1. Redes tecnológicas (technological networks)	45
3.13.2. Redes de información (information networks)	45
3.13.3. Redes sociales (social networks)	46
3.13.4. Redes biológicas (biological networks)	46
3.13.5. Ciudades inteligentes (smart cities)	47
3.13.6. Tipos de datos que representan grafos	48
3.13.7. Teoría de grafos	49
3.13.8. Redes de mundo pequeño (small world networks)	49
3.13.9. Redes libres de escala (scale free networks)	50
3.13.10. Componente gigante	50
3.13.11. Distancia geodésica	50
3.13.12. Medidas de centralidad	50
3.13.12.1. Centralidad de grado o grado nodal (Degree centrality)	50
3.13.12.2. Centralidad de intermediación (Betweenness centrality)	51
3.13.12.3. Centralidad de vector propio o autovector (Eigenvector centrality)	51
3.13.12.4. Centralidad de Bonacich	52
3.13.13. Detección de comunidades	52
3.13.14. Principales V's de Big Data para grafos	53
3.14. Bulk Synchronous Parallel (BSP)	54

3.14.1. Pregel	55
3.15. Sistemas para grafos (Graph systems)	56
3.15.1. Sistemas de almacenamiento	56
3.15.1.1. Neo4j	56
3.15.1.2. Titan	57
3.15.1.3. HDFS	57
3.15.1.4. DEX/Sparksee	57
3.15.1.5. Oracle Spatial	58
3.15.2. Sistemas de procesamiento	58
3.15.2.1. Ringo	58
3.15.2.2. Pegasus	59
3.15.2.3. Neo4j Mazerunner	59
3.15.2.4. Graphlab/PowerGraph	60
3.15.2.5. Apache Giraph	60
3.15.2.6. GraphX	60
3.16. Herramientas para la visualización de grafos	61
3.16.1. Gephi	61
3.16.2. Graphviz	61
3.16.3. D3.js	62
3.16.4. GraphStream	62
4. GraphX	63
4.1. Particionamiento de grafos de gran escala	64
4.2. Grafos o datos en paralelo. (Graph parallel or data parallel)	64
4.3. Tipos de flujos de datos	65
4.4. Almacenamiento de grafos	65
4.5. Algoritmos para grafos	65
4.5.1. PageRank	65
4.5.2. Contador de triángulos (Triangle Count)	66
4.5.3. Caminos cortos (Shortest Paths)	66
4.5.4. Componentes conectados (Connected Components)	67
4.5.5. Componentes fuertemente conectados (Strongly Connected Components)	67
4.5.6. Propagación por etiqueta (Label Propagation)	68
4.5.7. Dijkstra	69
4.5.8. Problema del agente viajero (Travelling Salesman)	69
5. Marco metodológico	70
6. Marco aplicativo: Sandbox Cloudera	74
6.1. Instalación de un solo nodo para pruebas (Sandbox Cloudera)	74
6.2. Entorno	76
6.3. Conjunto de datos de prueba (datasets)	77
6.4. Implementación	78
6.5. Pruebas	78
6.6. Resultados	79

7. Marco aplicativo: Clúster Hadoop	81
7.1. Clúster	81
7.2. Instalación del sistema operativo CentOS 7	81
7.3. Configuraciones de nodos	82
7.3.1. Modificación del archivo host	82
7.3.1.1. Creación de un hostname	82
7.3.1.1.1. Nodo 1 (node1)	82
7.3.1.1.2. Nodo 2 (node2)	83
7.3.1.1.3. Nodo 3 (node3)	83
7.3.1.1.4. Nodo Maestro (nodemaster)	83
7.3.1.2. Configuración del archivo network	83
7.3.1.2.1. Nodo 1 (node1)	83
7.3.1.2.2. Nodo 2 (node2)	83
7.3.1.2.3. Nodo 3 (node3)	83
7.3.1.2.4. Nodo Maestro (nodemaster)	83
7.3.1.3. Desactivación del servicio SELINUX	84
7.3.1.4. Configuración del Network Time Protocol (NTP) y desactivación del Firewall	84
7.3.1.5. Configuración del SSH	84
7.4. Instalación Cloudera Manager	85
7.5. Instalación de SBT	89
7.6. Extracción de datos	89
7.7. Implementación utilizando GraphX	90
7.7.1. Bibliotecas utilizadas	90
7.7.2. Configuraciones de Spark	91
7.7.3. Importar nodos y vértices	91
7.7.4. Contar nodos y vértices	91
7.7.5. Calcular grados de los nodos	91
7.7.6. Shortest Paths	92
7.7.7. Triangle Count	92
7.7.8. PageRank	93
7.7.9. Connected Components	93
7.7.10. Generar .gexf	93
7.8. Almacenamiento de los conjuntos de datos en HDFS	94
7.9. Extraer repositorio GitHub	94
7.10. Pruebas	95
7.11. Resultados	97
7.11.1. ShortestPaths	98
7.11.2. PageRank	99
7.11.3. TriangleCount	100
7.11.4. ConnectedComponents	101
7.11.5. Gephi	102
7.11.6. Descripcion	103

8. Marco aplicativo: Gephi	104
8.1. Importar grafo	104
8.2. Detección de comunidades	105
8.3. Selección de distribución	107
8.4. Cálculo de grados	109
8.5. PageRank	110
9. Conclusiones	112
9.1. Contribución	113
9.2. Recomendaciones	113
9.3. Trabajos Futuros	114
Anexos	115
9.4. Implementación realizada en Scala junto a GraphX	115
9.5. Encontrar inconvenientes del proceso de instalación de Cloudera	121
9.6. Desinstalar la distribución Cloudera	121

Índice de figuras

3.1.	Proceso para la minería de datos.	7
3.2.	Apache Hadoop.	22
3.3.	Arquitectura de HDFS.	24
3.4.	Arquitectura de YARN.	26
3.5.	Ejemplo de MapReduce.	27
3.6.	Arquitectura Apache Accumulo.	29
3.7.	Modelo utilizando Apache HBase.	30
3.8.	Arquitectura Apache Hive.	31
3.9.	Apache Spark.	37
3.10.	Conjuntos Distribuidos Resilientes.	39
3.11.	Apache Hadoop vs Apache Spark	41
3.12.	Red tecnológica. Red de metro.	45
3.13.	Red de información. Cicitación de revistas.	45
3.14.	Red social. Colaboración científica.	46
3.15.	Red biológica. Cadena trófica.	47
3.16.	Ciudades inteligentes.	48
3.17.	Red y árbol.	48
3.18.	RDBMS y matriz dispersa.	49
3.19.	Centralidad de intermediación.	51
3.20.	Bulk Synchronous Parallel.	54
3.21.	Pregel.	55
4.1.	Grafos o datos en paralelo.	64
4.2.	Almacenamiento de grafos.	65
4.3.	Shortest Paths.	67
4.4.	Connected Components.	67
4.5.	Strongly Connected Components.	68
4.6.	Label Propagation.	68
4.7.	Dijkstra.	69
5.1.	Metodología para Ciencia de los Datos.	70
6.1.	Descarga e instalación de VMware Workstation Pro 12.	75
6.2.	Descarga de Cloudera Quickstart VM CDH 5.7 Parte I.	75
6.3.	Descarga de Cloudera Quickstart VM CDH 5.7 Parte II.	75
6.4.	Abrir una máquina virtual.	76

6.5. Seleccionar una máquina virtual.	76
6.6. Conjunto de archivos utilizados.	76
6.7. Bibliotecas utilizadas.	77
6.8. Datasets.	77
6.9. Almacenar datos en HDFS.	78
6.10. Visualizar datos en HDFS.	78
6.11. Ejecución de Apache Spark y otros.	78
6.12. Ejecutando Spark Shell.	78
6.13. Importando componentes.	79
6.14. Código implementado en Sandbox Cloudera.	79
6.15. Código para calcular el número de vértices y aristas.	79
6.16. Resultado número de vértices y aristas del dataset de Facebook.	80
6.17. Algoritmos para calcular el vértice y su total de aristas que posea mayor InDegree y OutDegree.	80
6.18. Resultados InDegree y OutDegree del dataset de Facebook.	80
6.19. Visualización del dataset de Facebook utilizando GraphStream.	80
7.1. Cloudera Enterprise Data Hub Edition Trial.	86
7.2. Selección de hosts para el CDH clúster.	86
7.3. Instalaciones y configuraciones para el clúster Cloudera.	87
7.4. Instalación del CDH.	87
7.5. Selección del Core con Spark.	87
7.6. Proporcionar credenciales de inicio de sesión de SSH.	88
7.7. Distribución de los roles para HDFS.	88
7.8. Distribución de los roles para Spark y YARN.	88
7.9. Cloudera Manager.	89
7.10. Ego network.	90
7.11. Ejemplo de formato de listas de aristas.	91
7.12. Spark modo clúster.	95
7.13. Vista previa del resultado de ShortestPaths.	98
7.14. Información de ejecución del módulo ShortestPaths en modo YARN clúster.	98
7.15. Información de ejecución del módulo ShortestPath en modo YARN client.	98
7.16. Vista previa del resultado de PageRank.	99
7.17. Información de ejecución del módulo PageRank en modo YARN cluster.	99
7.18. Información de ejecución del módulo PageRank en modo YARN client.	99
7.19. Vista previa del resultado de TriangleCount.	100
7.20. Información de ejecución del módulo TriangleCount en modo YARN cluster.	100
7.21. Información de ejecución del módulo TriangleCount en modo YARN client.	100
7.22. Vista previa del resultado de ConnectedComponents.	101
7.23. Información de ejecución del módulo ConnectedComponents en modo YARN cluster.	101

7.24. Información de ejecución del módulo ConnectedComponents en modo YARN client.	101
7.25. Vista previa del resultado de Gephi.	102
7.26. Información de ejecución del módulo Gephi en modo YARN cluster.	102
7.27. Información de ejecución del módulo Gephi en modo YARN client.	102
7.28. Vista previa del resultado de Descripcion.	103
7.29. Información de ejecución del módulo Descripcion en modo YARN cluster.	103
7.30. Información de ejecución del módulo Descripcion en modo YARN client.	103
8.1. Importar grafo a Gephi.	104
8.2. Visualización inicial del grafo.	105
8.3. Ejecutar modularidad en Gephi.	105
8.4. Distribución de modularidad.	106
8.5. Coloración de las agrupaciones.	107
8.6. Distribución ForceAtlas 2.	107
8.7. Cambiar colores de las comunidades.	108
8.8. Distribución Fruchterman Reingold.	108
8.9. Distribución de los grados.	109
8.10. Distribución de los grados de entrada.	109
8.11. Distribución de los grados de salida.	110
8.12. Ejecutar algoritmo de PageRank en Gephi.	110
8.13. PageRank en Gephi.	111
8.14. Distribución del PageRank.	111
9.1. Comparación entre los tiempos de ejecución (YARN cluster vs YARN client).	112
9.2. Implementación realizada en Scala junto a GraphX. Parte I.	115
9.3. Implementación realizada en Scala junto a GraphX. Parte II.	116
9.4. Implementación realizada en Scala junto a GraphX. Parte III.	117
9.5. Implementación realizada en Scala junto a GraphX. Parte IV.	118
9.6. Implementación realizada en Scala junto a GraphX. Parte V.	119
9.7. Implementación realizada en Scala junto a GraphX. Parte VI.	119
9.8. Implementación realizada en Scala junto a GraphX. Parte VII.	120
9.9. Implementación realizada en Scala junto a GraphX. Parte VIII.	120

Índice de tablas

3.1.	Tipos de Base de Datos NoSql.	15
3.2.	Comparación entre HBase y HDFS.	31
3.3.	Algoritmos Mahout.	33
6.1.	Descripción del equipo donde se encuentra instalado el Sandbox Cloudera.	74
7.1.	Descripción de las máquinas utilizadas.	81
7.2.	Descripción de red y memoria.	81
7.3.	Dirección IPv4 de los nodos.	82
7.4.	Datasets utilizados.	89
7.5.	Módulos de pruebas.	95

CAPÍTULO 1

Introducción

A medida que pasa el tiempo algunos de los problemas en la computación cambian gracias a los avances científicos, en el pasado uno de los problemas era el almacenamiento y procesamiento de grandes volúmenes de datos (datos que no pueden ser manipulados por herramientas y procesos tradicionales), en el presente el problema es detectar oportunidades o valor en estos datos, en consecuencia las empresas cada vez se preocupan más en la obtención y recolección de datos. Las propiedades de los datos han evolucionado, donde el volumen de estos ha crecido, ha incrementado su diversidad en forma y origen, es decir la variedad, ha aumentado la necesidad de obtener resultados en tiempo real, en otras palabras la velocidad. La capacidad de generar buenas visualizaciones de los datos es una propiedad importante para las empresas ya que facilita la obtención de valor, lo que permite obtener criterios para la toma de decisiones. La gestión de los datos con las características mencionadas anteriormente, se consigue mediante los sistemas Big Data.

En la actualidad las redes se encuentran por todas partes, nos rodean y formamos parte de ellas, por lo tanto analizarlas puede ser de gran utilidad. Las redes de comunicación, la World Wide Web (www), las redes de proteínas, el genoma humano, las redes neuronales, las redes de transporte, las redes sociales, el lenguaje, las redes de colaboración científica o las redes terroristas son algunos ejemplos. En base a una estructura de grafo se puede representar una red y realizar análisis es relativamente sencillo ya que es posible aplicar conceptos de la teoría de grafos, la cual es una disciplina que posee muchos avances y tiene años siendo estudiada.

Muchas redes son un ejemplo claro de grandes volúmenes de datos, las redes sociales es una de estas y realizar distintos tipos de análisis puede ser de gran utilidad. Muchas organizaciones se han dado cuenta que la gestión y un análisis completo de los datos puede influir positivamente en la empresa si se realiza de forma adecuada y se toman las decisiones correctas. Combinando una estructura de grafo con conceptos de la teoría de grafos junto a Big Data es posible realizar análisis sobre redes de gran escala, es decir que representen un problema de grandes volúmenes de datos.

CAPÍTULO 2

Problema de investigación

2.1. Planteamiento del problema

Hoy en día, existen muchas organizaciones que generan muchos datos en corto tiempo y no son capaces de gestionarlos utilizando herramientas tradicionales (aquellas que no pueden procesar grandes volúmenes de datos). Desafortunadamente, las herramientas tradicionales, por ejemplo las base de datos relacionales, no son adecuadas para resolver el problema de grandes volúmenes de datos. Actualmente existen empresas que no poseen el problema anteriormente mencionado ya que utilizan herramientas Big Data para procesar sus datos, sin embargo existen estructuras que no son sencillas de manipular mediante estos sistemas, por ejemplo las redes. Gracias a la teoría de grafos podemos modelar y analizar las redes, lo cual facilita la realización de análisis.

Utilizando las herramientas tradicionales para el análisis de datos y los sistemas Big Data comunes de la actualidad, no es sencillo manipular datos que representen redes, ya que los métodos de procesamiento fallan cuando el tamaño de estos datos incrementa y la necesidad de realizar análisis en tiempo real aumenta. Hadoop es bueno ejecutando consultas simples sobre un gran conjunto de datos, pero en la vida real comúnmente es necesario ejecutar consultas complejas o iterativas (ver más en el capítulo 3.12). Implementar algoritmos iterativos con Hadoop MapReduce requiere una serie de operaciones en disco, donde los datos deben ser cargados nuevamente en cada iteración, lo que produce tiempos de ejecución muy largos.

Los grafos en la mayoría de los casos no representan un gran volumen de datos inicialmente, sin embargo a medida que el tamaño de este crece linealmente, aumenta exponencialmente en complejidad, es decir que su principal problema es de procesamiento. Existen herramientas especializadas que combinan la teoría de grafos y Big Data, donde la carga de datos se realiza en memoria. Estos sistemas proveen estructuras de almacenamiento, métodos de particionamiento, procesamiento y visualización para grafos, con interfaces, un ejemplo de esto es Apache Spark junto con GraphX. El objetivo es solventar el problema de procesamiento que posee Hadoop MapReduce, utilizando herramientas basadas en memoria, como Apache Spark.

2.2. Justificación

2.2.1. ¿Por qué es un problema?

Es un problema procesar redes de gran escala utilizando herramientas tradicionales y las herramientas Big Data comúnmente usadas en el mercado actual (Julio, 2016), como lo es Hadoop MapReduce. Hadoop MapReduce solventa el problema de procesar gran volúmenes de datos, sin embargo su procesamiento es alterado negativamente por algoritmos iterativos, debido a que realiza accesos a disco para cargar los datos, lo que trae como consecuencia tiempos de ejecución de mucha duración.

2.2.2. ¿Para quién es un problema?

Es un posible problema para cualquier usuario u organización que le interese y/o necesite realizar análisis de redes que representen un gran volumen de datos.

2.2.3. ¿Desde cuándo es un problema?

Es un problema desde que existen redes de gran tamaño, es decir que representadas como grafos contienen muchos nodos y aristas, y a su vez los nodos están altamente interconectados entre ellos.

2.3. Objetivos de la investigación

2.3.1. General

Manipular, almacenar, estructurar, visualizar, desarrollar cálculos y algoritmos basados en la teoría de grafos que permitan realizar análisis sobre diversas redes que representan un problema de grandes volúmenes de datos, utilizando herramientas especializadas en procesamiento paralelo y distribuido para grafos de gran escala.

2.3.2. Específicos

- Definir las herramientas a utilizar.
- Buscar datos que representen redes de gran escala.
- Almacenar y estructurar las redes en formatos específicos de tal forma que representen un grafo.
- Utilizar Apache Spark para procesar los diversos grafos y tener acceso a la utilización de HDFS y GraphX.
- Almacenar los grafos en HDFS.

- Extraer los grafos de HDFS y adaptarlos para su procesamiento utilizando GraphX.
- Implementar diversos cálculos para obtener información sobre los grafos.
- Implementar algoritmos que provean información compleja sobre los grafos.
- Representar gráficamente los grafos utilizando Gephi y GraphStream.
- Realizar pruebas de la aplicación.
- Visualizar los resultados obtenidos.

2.4. Antecedentes

Hadoop resuelve el problema de grandes volúmenes de datos proporcionando un marco de trabajo (framework) que permite realizar procesamientos en paralelos con tolerancia a fallos, utilizando clusters. Algunas de sus funciones principales son el almacenamiento distribuido utilizando HDFS y el procesamiento distribuido mediante MapReduce. MapReduce es un paradigma limitado, pero se ha utilizado para resolver muchos problemas de datos paralelos (ver más en el capítulo 3.12.4). Hadoop MapReduce tiene problemas con muchos algoritmos iterativos, por ejemplo PageRank, por lo tanto nace Spark.

Bulk Synchronous Parallel (BSP) (ver más en el capítulo 3.14) no fue diseñado específicamente para el procesamiento de grafos, pero cuando Google implementó su entorno de procesamiento de grafos, Pregel, utilizó los principios de BSP. Google Pregel fue la inspiración para Spark para desarrollar GraphX. La mayoría de los algoritmos que provee GraphX se implementaron en términos de Pregel.

2.5. Alcance

El presente trabajo de investigación se desarrollará una serie de operaciones y algoritmos que permitan la extracción, carga, almacenamiento, procesamiento, visualización y análisis de grafos de gran escala utilizando Apache Spark, HDFS y GraphX.

La realización de estas operaciones y algoritmos implica instalar las herramientas necesarias para su desarrollo y procesamiento. Las operaciones a implementar permitirán el almacenar, integrar y realizar consultas simples sobre los diversos grafos. Los algoritmos a desarrollar serán capaces de proveer información compleja y visualizar los distintos grafos.

2.6. Consideraciones

La propuesta planteada es la siguiente: ¿Es posible almacenar, procesar diversas operaciones y algoritmos, visualizar y analizar grafos de gran escala utilizando herramientas y/o utilidades referidas al área de la Ciencia de los Datos?

¿Qué se va a hacer?

Buscar y extraer una red de cualquier tipo (ver más a partir del capítulo 3.13.1) que represente un problema de grandes volúmenes de datos, estructurar la red como un grafo, almacenar el grafo en algún sistema de almacenamiento distribuido, procesar el grafo con algún sistema de procesamiento paralelo, realizar operaciones y algoritmos utilizando algún lenguaje de programación, y visualizar el grafo mediante alguna herramienta especializada en redes de gran escala.

¿Cómo se va a hacer?

Utilizando sistemas de almacenamiento para grafos, sistemas de procesamiento de grandes volúmenes de datos, lenguajes de programación y APIs que faciliten la realización de diversas operaciones y algoritmos basados en la teoría de grafos (ver más en el capítulo 3.13.7) y herramientas de visualización especializadas en redes de gran tamaño.

2.6.1. Metodología

Se utilizará la metodología fundamental para la Ciencia de Datos propuesta por la empresa IBM (ver más capítulo 5).

2.6.2. Tecnologías a utilizar

- Apache Hadoop.
- Distribución Hadoop: Cloudera.
- HDFS.
- Apache Spark.
- GraphX.
- Scala.
- GraphStream.
- Gephi.

CAPÍTULO 3

Marco teórico

3.1. Dato

Los datos son números, letras o símbolos que describen objetos, condiciones o situaciones. Son el conjunto básico de hechos referentes a una persona, cosa o transacción de interés para distintos objetivos, entre los cuales se encuentra la toma de decisiones. [1]

3.2. Información

La información es un sistema de control, en tanto que es la propagación de consignas que deberíamos de creer. En tal sentido la información es un conjunto organizado de datos capaz de cambiar el estado de conocimiento. Un grupo de datos ordenados y supervisados, que sirven para construir un mensaje basado en un cierto fenómeno o ente, la cual permite resolver problemas y tomar decisiones, es información, ya que su aprovechamiento racional es la base del conocimiento. [2]

3.3. Conocimiento

Fidias Arias (2004), define el conocimiento como un "proceso en el cual se relacionan el sujeto que conoce, que percibe mediante sus sentidos, y el objeto conocido y percibido". El conocimiento es el acto o efecto de conocer. Es la capacidad del hombre para comprender por medio de la razón la naturaleza, cualidades y relaciones de las cosas.

La ciencia considera que, para alcanzar el conocimiento, es necesario seguir un método. El conocimiento científico no sólo debe ser válido y consistente desde el punto de vista lógico, sino que también debe ser probado mediante el método científico o experimental. [3]

3.4. Minería de datos (Data Mining)

Es un campo de la estadística y las ciencias de la computación que tiene como objetivo detectar la información procesable o patrones sobre conjuntos grandes de datos. Normalmente, estos patrones no se pueden detectar mediante la exploración tradicional de los datos porque las relaciones son demasiado complejas o porque hay demasiados datos. [4]

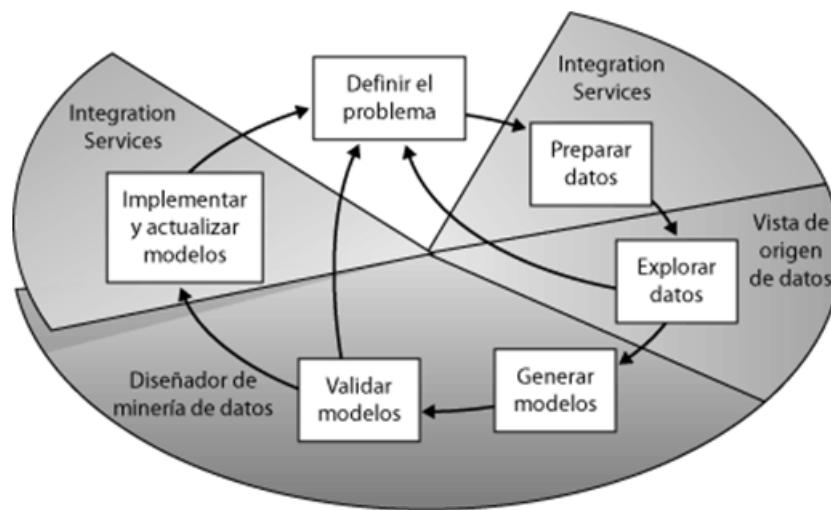


Figura 3.1: Proceso para la minería de datos.

La generación de un modelo de minería de datos forma parte de un proceso mayor que incluye desde la formulación de preguntas acerca de los datos y la creación de un modelo para responderlas, hasta la implementación del modelo en un entorno de trabajo. Este proceso se puede definir mediante los seis pasos básicos siguientes:

- Definir el problema: Este paso incluye analizar los requisitos organizacionales, definir el ámbito del problema, definir las métricas por las que se evaluará el modelo y definir los objetivos concretos del proyecto de minería de datos.
- Preparar los datos o preprocesamiento: Los datos pueden estar dispersos en la empresa y almacenados en formatos distintos; también pueden contener incoherencias como entradas que faltan o incorrectas. La finalidad de este paso es consolidar y limpiar los datos.
- Explorar los datos o análisis exploratorio: Entre las técnicas de exploración se incluyen calcular los valores mínimos y máximos, calcular la media y las desviaciones estándar, y examinar la distribución de los datos. Al explorar los datos se puede decidir si el conjunto contiene datos defectuosos y, a continuación, puede proponer una estrategia para corregir los problemas u obtener una descripción más profunda de los comportamientos que son típicos de su negocio.

- Generar modelos: Consiste en generar uno o varios modelos de minería de datos. Antes de procesar la estructura y el modelo, un modelo de minería de datos simplemente es un contenedor que especifica las columnas que se usan para la entrada, el atributo que está prediciendo y parámetros que indican al algoritmo cómo procesar los datos. El procesamiento de un modelo a menudo se denomina entrenamiento.
- Validar modelos: Consiste en explorar los modelos de minería de datos que ha generado y comprobar su eficacia. Antes de implementar un modelo en un entorno de producción, es aconsejable probar si funciona correctamente.
- Implementar y actualizar los modelos: Consiste en implementar los modelos que funcionan mejor en un entorno de producción.

3.4.1. Minería de grafos (Graph Mining)

Minería de grafos es un área de la minería de datos que tiene como objetivo encontrar patrones, asociaciones y anomalías en los grafos. Los grafos o redes están por todas partes, van desde grafos Web, redes sociales, redes biológicas, y muchos más. Encontrar patrones, reglas y anomalías en los grafos tienen numerosas aplicaciones, por ejemplo:

- Ranking de páginas web para un motor de búsqueda.
- Marketing viral (Ej: Pokemon Go).
- Patrones en las enfermedades.
- Red de seguridad: tráfico de correo electrónico y la detección de anomalías.

3.5. Aprendizaje automático (Machine Learning)

En ciencias de la computación el aprendizaje automático es una rama de la inteligencia artificial cuyo objetivo es desarrollar técnicas que permitan a las computadoras aprender. De forma más concreta, se trata de crear programas capaces de generalizar comportamientos a partir de una información no estructurada suministrada en forma de ejemplos. El aprendizaje automático se divide en dos áreas principales: aprendizaje supervisado y aprendizaje no supervisado.[5]

El objetivo del aprendizaje supervisado es hacer predicciones a futuro basadas en comportamientos o características que se han visto en los datos ya almacenados. El aprendizaje supervisado permite buscar patrones en datos históricos etiquetados. Por otro lado, el aprendizaje no supervisado usa datos históricos que no están etiquetados. El fin es explorarlos para encontrar alguna estructura o forma de organizarlos.

3.6. Inteligencia artificial (Artificial Intelligence)

Inteligencia artificial es considerada una rama de la computación que relaciona un fenómeno natural con una analogía artificial mediante programas de computador. Consiste en el diseño de procesos que, al ejecutarse sobre una arquitectura física, producen resultados que maximizan una cierta medida de rendimiento denominado comúnmente comportamiento inteligente. [6]

3.7. Inteligencia de negocios (Business Intelligence)

Inteligencia de negocios es el conjunto de conceptos y métodos para mejorar la toma de decisiones en los negocios, utilizando sistemas de apoyo basados en hechos (Howard Dresner, 1989). Sin embargo, en la actualidad este concepto incluye una amplia categoría de metodologías, aplicaciones y tecnologías que permiten reunir, acceder, transformar y analizar los datos con el propósito de ayudar a los usuarios de una organización a tomar mejores decisiones de negocio [7]

3.8. Ciencia de datos

Ciencias de los datos es un campo interdisciplinario, el cual abarca un conjunto de procesos y sistemas para extraer información de un conjunto de datos estructurados o no estructurados con el objetivo de generar conocimiento. Esta área está estrechamente relacionada con las estadísticas, la minería de datos, análisis predictivo, entre otros. La ciencia de datos utiliza la preparación de datos, estadísticas, modelos predictivos y de aprendizaje automático para investigar problemas en diversos ámbitos. Múltiples personas en el mercado actual utilizan el concepto Big Data igual que el de Ciencia de datos pero haciendo énfasis a grandes volúmenes de datos.[8]

La ciencia de datos permite la generación de conocimiento a partir de grandes volúmenes de datos, aplicando técnicas de procesamiento paralelo y distribuido, para implementar algoritmos que permitan predecir o detectar patrones sobre los datos almacenados. A partir de los resultados obtenidos se podrán construir herramientas que permitan analizar los resultados y apoyar los procesos de toma de decisiones.

3.8.1. Big data

Es una plataforma tecnológica (hardware y software) que permite almacenar (de manera distribuida) y procesar (en forma paralela y distribuida) conjuntos de datos, que por su gran volumen, superan las capacidades de las plataformas TI tradicionales, ya sea porque tomaría demasiado tiempo procesar dichos datos o porque sería muy costoso implementar una arquitectura que soporte tal cantidad de datos.

En otras palabras, Big data es un concepto que hace referencia a la acumulación de grandes cantidades de datos y a los procedimientos usados para encontrar patrones repetitivos dentro de estos. Se define como un conjunto de herramientas informáticas destinadas a la manipulación, gestión y análisis de grandes volúmenes de datos de todo tipo, los cuales no pueden ser gestionados por las herramientas informáticas tradicionales, que tiene por objetivo analizar datos e información de manera inteligente que ayuden a una correcta toma de decisión.

En la actualidad la cantidad de información digital que se genera diariamente en nuestro planeta crece exponencialmente, lo cual ha desencadenado que muchas empresas y organizaciones, deseen utilizar esta información con el objetivo de mejorar la prestación de sus servicios o negocios. Por lo tanto, el objetivo fundamental de los sistemas Big Data es dotar de una infraestructura tecnológica a las empresas y organizaciones con la finalidad de poder almacenar, tratar y analizar de manera económica, rápida y flexible la gran cantidad de datos que se generan diariamente, para ello es necesario el desarrollo y la implantación tanto de hardware como de software específicos que gestionen esta explosión de datos con el objetivo de extraer valor para obtener información útil para nuestros objetivos o negocios. [9]

El término de Big Data, en general se definen con ocho dimensiones llamada las 8V [10], Velocidad, Veracidad, Valor, Volumen, Variedad, Variabilidad, Visualización y Visión:

- **Velocidad:** La tecnología Big Data ha de ser capaz de almacenar y trabajar en tiempo real con las fuentes generadoras de información como sensores, cámaras de vídeos, redes sociales, blogs, páginas webs, entre otros, es decir fuentes que generan millones y millones de datos al segundo, por otro lado la capacidad de análisis de dichos datos han de ser rápidos reduciendo los largos tiempos de procesamiento que presentaban las herramientas tradicionales de análisis.
- **Veracidad:** Big Data ha de ser capaz de tratar y analizar inteligentemente este vasto volumen de datos con la finalidad de obtener una información verídica y útil que permita mejorar la toma de decisiones.
- **Volumen de datos:** Como su propio nombre indica la tecnología Big Data ha de ser capaz de gestionar el gran volumen de datos que se generan diariamente por las empresas y organizaciones de todo el mundo.
- **Variedad de datos:** Big data ha de tener la capacidad de combinar una gran variedad de información digital en los diferentes formatos en las que se puedan presentar ya sean en formato vídeo, audio, texto u otro.

- Valor: El valor se refiere a la capacidad de convertir los datos en valor. Es importante para las empresas realizar cualquier intento de recoger y aprovechar los datos masivos. Mientras más datos disponibles la probabilidad de obtener valor aumenta. El valor está en los análisis realizados en los datos y cómo estos se transforman en información, y finalmente, convirtiéndola en conocimiento. El valor está en cómo las organizaciones usarán el conocimiento obtenido para tomar decisiones.
- Variabilidad: La variabilidad se refiere a los datos cuyo significado está en constante cambio. Este es particularmente el caso cuando la recolección de datos se basa en el procesamiento del lenguaje, donde el significado de las palabras puede cambiar con el tiempo y por la ubicación geográfica.
- Visualización: Esta es la parte dura de Big Data, hacer que la gran cantidad de datos sea comprensible de manera que sea fácil de entender y leer. Con los análisis y visualizaciones adecuadas, los datos brutos se pueden utilizar para tomas de decisiones adecuadas. Las visualizaciones por supuesto no significan gráficas ordinarias o gráficos circulares. Significan gráficos complejos que pueden incluir muchas variables de datos sin dejar de ser comprensible y legible. La visualización puede no ser la parte más difícil con respecto al uso de la tecnología, pero lograr buenos resultados es complicado. Contar una historia compleja en un gráfico es muy difícil, pero también es extremadamente crucial.
- Visión: todas las empresas, que se inician con grandes volúmenes de datos deben tener una visión, qué hacer con ellos. La visión define las metas que se pretenden conseguir en el futuro. Estas metas tienen que ser realistas y alcanzables, puesto que la propuesta de visión tiene un carácter inspirador y motivador. La descarga de Hadoop, la instalación de él y la alimentación con algunos datos no ayudarán. La empresa tiene que estar lista para la transformación digital. Si la organización no entiende lo que puede ofrecer grandes volúmenes de datos, no habrá ningún éxito.

3.8.2. Campos en los cuales es utilizado

El manejo de grandes volúmenes de datos es utilizado y se puede utilizar en múltiples áreas, por ejemplo:

- Seguridad: Su potencial reside en la capacidad de análisis de volúmenes de datos antes impensable de una manera óptima y ágil. Existen, por ejemplo, modelos de análisis del comportamiento humano para prevenir atentados terroristas obtenidos mediante un análisis permanente de las cámaras, sensores y accesos secuenciales a un sistema.
- Investigación médica: La investigación médica puede mejorar muchísimo si es capaz de asimilar una enorme cantidad de datos (monitorización, historiales,

tratamientos, etc.) y estructurarlos para el establecimiento de diagnósticos o la síntesis de medicamentos.

- Gobierno y toma de decisiones: Big Data ofrece una mejora y optimización en los procesos de análisis de empresas y gobiernos, permitiendo entre muchas otras, el soporte a la toma de decisiones, siendo complementario a las plataformas de "Business Intelligence" (BI).
- Internet 2.0: genera una gran multitud de datos que difícilmente se podrían gestionar sin un Big Data. Las redes sociales cada vez se extienden a más ámbitos de nuestra sociedad
- Logística: El sector logístico mejora notablemente gracias a las posibilidades analíticas de un Big Data y su potencial para el despliegue de servicios específicos (movilidad, tracking, seguridad, etc.). El ejemplo más popular se encuentra en el control de flotas (la ruta óptima permite a los vehículos circular con la máxima capacidad de carga, pudiendo recorrer rutas mejorando tiempos, consumos y contaminación).

3.9. Base de datos

Se define una base de datos como una serie de datos organizados y relacionados entre sí, los cuales son recolectados y explotados por los sistemas de información de una empresa o negocio en particular. [11] Entre las principales características de los sistemas de base de datos podemos mencionar:

- Independencia lógica y física de los datos.
- Redundancia mínima.
- Acceso concurrente por parte de múltiples usuarios.
- Integridad de los datos.
- Consultas complejas optimizadas.
- Seguridad de acceso y auditoría.
- Respaldo y recuperación.
- Acceso a través de lenguajes de programación estándar.

3.9.1. Base de datos relacionales

Una base de datos relacional es una colección de elementos de datos organizados en un conjunto de tablas formalmente descritas desde la que se puede acceder a los datos o volver a montarlos de muchas maneras diferentes sin tener que reorganizar las tablas de la base.

Estas bases de datos poseen un conjunto de tablas que contienen datos provistos en categorías predefinidas. Cada tabla (que a veces se llaman ‘relación’) contiene una o más categorías de datos en columnas. Cada fila contiene una instancia única de datos para las categorías definidas por las columnas. [12]

3.9.1.1. Ventajas de las base de datos relacionales

- Está más adaptado su uso y los perfiles que los conocen son mayoritarios y más baratos.
- Debido al largo tiempo que llevan en el mercado, estas herramientas tienen un mayor soporte y mejores suites de productos para gestionar estas bases de datos.
- La atomicidad de las operaciones en la base de datos.
- Los datos deben cumplir requisitos de integridad tanto en tipo de dato como en compatibilidad.

3.9.1.2. Desventajas de las base de datos relacionales

- La atomicidad de las operaciones juegan un papel crucial en el rendimiento de las bases de datos.
- Escalabilidad, que aunque probada en muchos entornos productivos suele, por norma, ser inferior a las bases de datos NoSQL.

3.9.2. Base de datos no relacionales

Son un enfoque hacia la gestión de datos y el diseño de base de datos que es útil para grandes conjuntos de datos distribuidos. Los datos almacenados no requieren estructuras fijas como tablas, normalmente no soportan operaciones JOIN, ni garantizan completamente ACID (atomicidad, consistencia, aislamiento y durabilidad), y habitualmente escalan bien horizontalmente.

Son especialmente útiles cuando una empresa necesita acceder y analizar grandes cantidades de datos no estructurados o datos que se almacenan de forma remota en varios servidores virtuales en la nube. [13]

3.9.2.1. Ventajas de las base de datos no relacionales

- La escalabilidad y su carácter descentralizado. Soportan estructuras distribuidas.
- Suelen ser bases de datos mucho más abiertos y flexibles. Permiten adaptarse a necesidades de proyectos más fácilmente que los modelos de Entidad Relación.
- Se pueden hacer cambios de los esquemas sin tener que detener las bases de datos.
- Escalabilidad horizontal: son capaces de crecer en número de máquinas, en lugar de tener que residir en grandes máquinas.
- Se pueden ejecutar en máquinas con pocos recursos.
- Optimización de consultas en base de datos para grandes cantidades de datos.

3.9.2.2. Desventajas de las base de datos no relacionales

- No todas las bases de datos NoSQL contemplan la atomicidad de las instrucciones y la integridad de los datos. Soportan lo que se llama consistencia eventual.
- Problemas de compatibilidad entre instrucciones SQL. Las nuevas bases de datos utilizan sus propias características en el lenguaje de consulta y no son 100 % compatibles con el SQL de las bases de datos relacionales. El soporte a problemas con las queries de trabajo en una base de datos NoSQL es más complicado.
- Falta de estandarización. Existen muchas bases de datos NoSQL y aún no hay un estándar como lo hay en las bases de datos relacionales. Se presume un futuro incierto en estas bases de datos.
- Soporte multiplataforma. Aún quedan muchas mejoras en algunos sistemas para que soporten sistemas operativos que no sean Linux.
- Suelen tener herramientas de administración no muy usables o se accede por consola.

3.9.2.3. Tipos de base de datos no relacionales

MODELO DE DATOS	FORMATO	CARACTERÍSTICAS	APLICACIONES
Documento.	Similar a JSON (JavaScript Object Notation).	<ul style="list-style-type: none"> - Intuitivo. - Manera natural de modelar datos cercana a la programación orientada a objetos. - Flexibles, con esquemas dinámicos. - Reducen la complejidad de acceso a los datos. 	Se pueden utilizar en diferentes tipos de aplicaciones debido a la flexibilidad que ofrecen.
Grafo.	Nodos con propiedades (atributos) y relaciones (aristas).	<ul style="list-style-type: none"> - Los datos se modelan como un conjunto de relaciones entre elementos específicos. - Flexibles, atributos y longitud de registros variables. - Permite consultas más amplias y jerárquicas. 	Redes sociales, software de recomendación, geolocalización, topologías de red, etc.
Clave-Valor y Columna.	Clave-valor: una clave y su valor correspondiente Columnas: variante que permite más de un valor (columna) por clave.	<ul style="list-style-type: none"> - Rendimiento muy alto. - Alta curva de escalabilidad. - Útil para representar datos no estructurados. 	Aplicaciones que solo utilizan consulta de datos por un solo valor de la clave.

Tabla 3.1: Tipos de Base de Datos NoSql.

3.9.2.3.1. Orientada a columnas

Este tipo de bases de datos están pensadas para realizar consultas y agregaciones sobre grandes cantidades de datos. Funcionan de forma parecida a las bases de datos relacionales, pero almacenando columnas de datos en lugar de registros. Algunos

ejemplos de base de datos orientada a columnas: Cassandra, HBase.

- Cassandra: incluida en esta sección, aunque en realidad sigue un modelo híbrido entre orientada a columnas y clave-valor. Es utilizada por Facebook y Twitter (aunque dejaron de usarla para almacenar tweets).
- HBase. Escrita en Java y mantenida por el Projecto Hadoop de Apache, se utiliza para procesar grandes cantidades de datos. La utilizan Facebook, Twitter o Yahoo.

3.9.2.3.2. Orientada a clave/valor

Son sencillas de entender. Simplemente guardan tuplas que contienen una clave y su valor. Cuándo se quiere recuperar un dato, simplemente se busca por su clave y se recupera el valor. Algunos ejemplos de base de datos clave/valor: DynamoDB, Redis.

- DynamoDB: desarrollada por Amazon, es una opción de almacenaje que podemos usar desde los Amazon Web Services. La utilizan el Washington Post y Scopely.
- Redis: desarrollada en C y de código abierto, es utilizada por Craigslist y Stack Overflow (a modo de caché).

3.9.2.3.3. Orientada a documentos

Son aquellas que gestionan datos semi estructurados, en otras palabras documentos. Estos datos son almacenados en algún formato estándar como puede ser XML, JSON o BSON. Son las bases de datos NoSQL más versátiles. Se pueden utilizar en gran cantidad de proyectos, incluyendo muchos que tradicionalmente funcionarían sobre bases de datos relacionales. Algunos ejemplos de base de datos orientada a documentos: MongoDB, CouchDB.

- MongoDB: probablemente la base de datos NoSQL más famosa del momento. En octubre del año pasado, MongoDB conseguía 150 millones de dólares en financiación, convirtiéndose en una de las startups más prometedoras. Algunas compañías que actualmente utilizan MongoDB son Foursquare o eBay.
- CouchDB: es la base de datos orientada a documentos de Apache. Una de sus interesantes características es que los datos son accesibles a través de una API Rest. Este sistema es utilizado por compañías como Credit Suisse y la BBC.

3.9.2.3.4. Orientada a grafo

Basadas en la teoría de grafos utilizan nodos y aristas para representar los datos almacenados. Son muy útiles para guardar información en modelos con muchas relaciones, como redes y conexiones sociales. Algunos ejemplos de base de datos orientada a grafos: Infinite Graph, Neo4j.

- Infinite Graph: escrita en Java y C++ por la compañía Objectivity. Tiene dos modelos de licenciamiento: uno gratuito y otro de pago.
- Neo4j: base de datos de código abierto, escrita en Java por la compañía Neo Technology. Utilizada por compañías como HP, Infojobs o Cisco.

3.10. Almacén de datos (Data Warehouse)

Un almacén de datos es una base de datos corporativa o repositorio de datos, que se caracteriza por integrar y depurar información de una o más fuentes distintas, para luego procesarla permitiendo su análisis desde infinidad de perspectivas y con grandes velocidades de respuesta. La creación de un almacén de datos colecciona datos, orientado a temas, integrado, no volátil, de tiempo variante el cual representa en la mayoría de las ocasiones el primer paso, desde el punto de vista técnico, para implantar una solución completa y fiable de inteligencia de negocios. Existen 2 investigadores muy famosos, Bill Inmon y Ralph Kimball relacionados con este concepto de almacén de datos. [14]

Para Bill Inmon (1992), quien acuñó el término por primera vez, “el Almacén de Datos es una colección de datos orientados al tema, integrados, no volátiles e históricos, organizados para el apoyo de un proceso de ayuda a la decisión. No obstante, y como cabe suponer, es mucho más que eso”.

Para Ralph Kimball (1997) “el Almacén de Datos es una copia de las transacciones de datos específicamente estructurada para la consulta y el análisis; es la unión de todos los Bodegas de Datos de una entidad”.

3.11. Lenguajes de programación

En computación, un lenguaje de programación es cualquier lenguaje artificial, que intenta conservar una similitud con el lenguaje humano, el cual, se utiliza para definir adecuadamente una secuencia de instrucciones que puedan ser interpretadas y ejecutadas en una computadora. [15]

Establecen un conjunto de símbolos, reglas sintácticas y semánticas, las cuales rigen la estructura y el significado del programa, junto con sus elementos y expresiones. De esta forma, permiten a los programadores o desarrolladores, poder especificar de forma precisa los datos sobre los que se va a actuar, su almacenamiento, transmisión y demás acciones a realizar bajo las distintas circunstancias consideradas. Usualmente se clasifican en interpretados y compilados, en el cual los compilados tienen un compilador específico que obtiene como entrada un programa y traduce las instrucciones las cuales pueden servir de entrada para otro interprete o compilado y los interpretados tienen un intérprete específico que obtiene como entrada un programa y ejecuta las acciones escritas a medida que las va procesando.

3.11.1. R

R es un lenguaje y entorno de programación para análisis estadístico y gráfico, el cual proporciona un amplio abanico de herramientas estadísticas (modelos lineales y no lineales, tests estadísticos, análisis de series temporales, algoritmos de clasificación y agrupamiento, etc.) y gráficas. Se trata de un proyecto de software libre, resultado de la implementación GNU del premiado lenguaje S y se distribuye bajo la licencia GNU GPL y está disponible para los sistemas operativos Windows, Macintosh, Unix y GNU/Linux. Puede integrarse con distintas bases de datos y existen bibliotecas que facilitan su utilización desde lenguajes de programación interpretados como Perl y Python. [16]

R al estar orientado a las estadísticas, proporciona un amplio abanico de herramientas de cálculo numérico y a su vez para minería de datos, y posee una capacidad gráfica, que permite generar gráficos con alta calidad, con sólo utilizar las funciones de graficación.

3.11.1.1. R Studio

RStudio es un entorno de desarrollo integrado (IDE) construido exclusivo para R, para computación estadística y gráficos . Incluye una consola, editor de sintaxis que apoya la ejecución de código, así como herramientas para el trazado, la depuración y la gestión del espacio de trabajo. [17]

Algunas de sus características son:

- Ejecutar código R directamente desde el editor de código fuente.
- Salto rápido a las funciones definidas.
- Colaborativo.
- Documentación y soporte integrado.
- Administración sencilla de múltiples directorios de trabajo mediante proyectos.
- Navegación en espacios de trabajo y visor de datos.
- Potente autoría y depuración.
- Depurador interactivo para diagnosticar y corregir los errores rápidamente.
- Herramientas de desarrollo extensas.
- Autoría con Sweave y R Markdown.

3.11.1.2. Igraph

Es un paquete que provee R que permite realizar análisis de redes. Proporciona rutinas y funciones para crear y manipular grafos con facilidad. En otras palabras, es una colección de herramientas para análisis de redes de forma eficiente, portable y de sencillo uso. Igraph es de código abierto y libre, y permite la utilización de los lenguajes R, Python, C y C++. [18]

3.11.2. Java

Java es un lenguaje de programación de propósito general, concurrente, orientado a objetos que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Su sintaxis deriva en gran medida de C y C++, pero tiene menos utilidades de bajo nivel que cualquiera de ellos. Las aplicaciones de Java son generalmente compiladas a bytecode (clase Java) que puede ejecutarse en cualquier máquina virtual Java (JVM) sin importar la arquitectura de la computadora subyacente.

Una de las principales características por las que Java se ha hecho muy famoso es que es un lenguaje independiente de la plataforma. Es una ventaja significativa para los desarrolladores de software, pues antes tenían que hacer un programa para cada sistema operativo, por ejemplo Windows, Linux, Apple, etc. Esto lo consigue porque se ha creado una Máquina de Java para cada sistema que hace de puente entre el sistema operativo y el programa de Java y posibilita que este último se entienda perfectamente. En la actualidad es un lenguaje muy extendido y cada vez cobra más importancia tanto en el ámbito de Internet como en la informática en general. [19]

El lenguaje Java se creó con cinco objetivos principales:

- Debería incluir por defecto soporte para trabajo en red.
- Debería usar el paradigma de la programación orientada a objetos.
- Debería permitir la ejecución de un mismo programa en múltiples sistemas operativos.
- Debería ser fácil de usar y tomar lo mejor de otros lenguajes orientados a objetos, como C++.
- Debería diseñarse para ejecutar código en sistemas remotos de forma segura.

3.11.3. Scala

El nombre de Scala significa "Scalable Languaje", se llama así ya que fue diseñado para poder crecer según la demanda de los usuarios, se puede usar Scala para crear pequeños scripts hasta para desarrollar grandes sistemas muy sofisticados. Scala es

un lenguaje de programación multi-paradigma que une la programación orientado a objetos y la programación funcional, que promueven la escalabilidad desde lo más pequeño. [20] Las principales características de Scala son:

- Una sintaxis concisa, elegante y flexibles: Scala utiliza una serie de técnicas para minimizar sintaxis innecesaria. La inferencia de tipos minimiza la necesidad de información de tipo explícito en muchos contextos. Las declaraciones de tipos y funciones son muy concisas. Scala permite incluir caracteres no alfanuméricos en los nombres de las funciones. Combinado con un poco de azúcar sintáctica, esta característica permite al usuario definir métodos que se parezcan y se comporten como operadores.
- Tipado estático: Scala es un lenguaje de tipado estático, lo que quiere decir que une el tipo a una variable durante toda la vida de esa variable.
- Orientado a Objetos: Scala es un lenguaje orientado a objetos puro, en el sentido de que cada valor es un objeto y cada operación es una llamada de método.
- Funcional: A pesar de que su sintaxis es bastante convencional, Scala es también un lenguaje funcional en toda regla. Tiene todo lo que se puede esperar, incluyendo funciones de primera clase, funciones anónimas, funciones de orden superior, funciones anidadas, una biblioteca con estructuras de datos inmutables eficientes, y una preferencia general de inmutabilidad sobre la mutabilidad.
- JVM (Java Virtual Machine): Scala se ejecuta en todas las máquinas virtuales de Java y también en Android. Las clases de Java y Scala pueden combinarse libremente, sin importar si residen en diferentes proyectos o en el mismo. Incluso pueden referirse mutuamente entre sí.
- Arquitecturas Escalables: Scala está diseñado para escalar desde pequeños scripts interpretados hasta grandes aplicaciones.

3.11.4. Python

Python es un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma, cuya filosofía hace hincapié en una sintaxis que favorezca un código legible. Es administrado por la Python Software Foundation. Posee una licencia de código abierto, denominada Python Software Foundation License. [21] Algunas de las características de Python son:

- Orientado a Objetos : La programación orientada a objetos está soportada en Python y ofrece en muchos casos una manera sencilla de crear programas con componentes reutilizables.

- Funciones y librerías: Dispone de muchas funciones incorporadas en el propio lenguaje, para el tratamiento de strings, números, archivos, etc. Además, existen muchas librerías que se pueden importar en los programas para tratar temas específicos como la programación de ventanas o sistemas en red o cosas tan interesantes como crear archivos comprimidos en .zip.
- Propósito general: Se pueden crear todo tipo de programas. No es un lenguaje creado específicamente para la web, aunque entre sus posibilidades sí se encuentra el desarrollo de páginas.
- Multiplataforma: Hay versiones disponibles de Python en muchos sistemas informáticos distintos. Originalmente se desarrolló para Unix, aunque cualquier sistema es compatible con el lenguaje siempre y cuando exista un intérprete programado para él.
- Interpretado: Quiere decir que no se debe compilar el código antes de su ejecución. En realidad sí se realiza una compilación, pero esta lo hace de manera transparente para el programador. En ciertos casos, cuando se ejecuta por primera vez un código, se producen unos bytecodes que se guardan en el sistema y que sirven para acelerar la compilación implícita que realiza el intérprete cada vez que se ejecuta el mismo código.
- Interactivo: Python dispone de un intérprete por línea de comandos en el que se pueden introducir sentencias. Cada sentencia se ejecuta y produce un resultado visible, que puede ayudar a entender mejor el lenguaje y probar los resultados de la ejecución de porciones de código rápidamente.
- Sintaxis clara: Por último, destacar que Python tiene una sintaxis muy visual, gracias a una notación identada (con márgenes) de obligado cumplimiento. En muchos lenguajes, para separar porciones de código, se utilizan elementos como las llaves o las palabras clave begin y end. Para separar las porciones de código en Python se debe tabular hacia dentro, colocando un margen al código que iría dentro de una función o un bucle. Esto ayuda a que todos los programadores adopten unas mismas notaciones y que los programas de cualquier persona tengan un aspecto muy similar.

3.12. Apache Hadoop

Apache Hadoop es un framework que permite el procesamiento de grandes volúmenes de datos a través de clusters, usando un modelo simple de programación. [22] En otras palabras, es un framework de software que soporta aplicaciones distribuidas bajo una licencia libre de la comunidad de Apache. Algunas de sus ventajas se basan en sus principales cualidades:

- Velocidad: garantiza una alta eficiencia de procesamiento.

- Flexibilidad: se adapta a las necesidades del negocio, permite la utilización de diversas fuentes de datos y distintos tipos de datos.
- Escalabilidad: permite almacenar y distribuir conjuntos de datos inmensos en sus cientos de servidores que operan en paralelo, permitiendo olvidarse de los límites que otras alternativas imponen.
- Resistencia al fracaso: su tolerancia a errores es uno de sus atributos mejor valorados por los usuarios ya que toda la información contenida en cada nodo tiene su réplica en otros nodos del cluster. En caso de producirse un fallo siempre existirá una copia lista para ser usada.

Hadoop usa una arquitectura maestro-esclavo (Master-Slave), usando para almacenar datos Hadoop Distributed File System (HDFS) y algoritmos de MapReduce para hacer cálculos. Permite a las aplicaciones trabajar con miles de nodos y petabytes de datos. Hadoop trata de ser confiable, proveer alta disponibilidad y manejo de fallos.

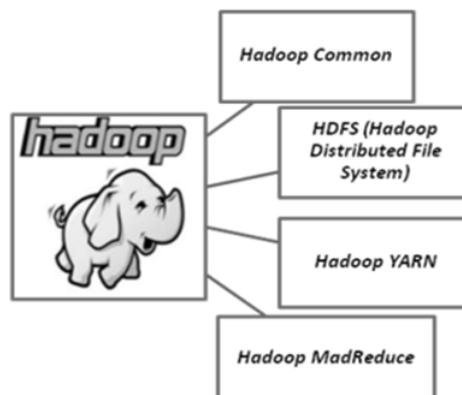


Figura 3.2: Apache Hadoop.

Los servicios de Hadoop proporcionan almacenamiento, procesamiento, acceso, gestión, seguridad y operaciones de datos. Posee 4 módulos esenciales, Common, HDFS, YARN y MapReduce.

3.12.1. Common

Contiene bibliotecas y otras utilidades que necesitan los otros módulos de Hadoop. Provee un conjunto de utilidades que permiten el soporte a otros proyectos de Hadoop, estas utilidades son consideradas como el núcleo del framework que provee servicios esenciales para otras aplicaciones basadas en Hadoop. [23] Proporciona abstracciones a nivel de sistema de archivos o sistema operativo. Contiene los archivos .jar (Java ARchive) y scripts necesarios para iniciar Hadoop. Al igual que todos los demás módulos, Hadoop Common maneja los fallos de hardware comunes de forma automática. El proyecto Apache Commons se compone de tres partes:

- Commons Proper: Un repositorio de componentes Java reutilizables.
- Commons Sandbox: Un espacio de trabajo para el desarrollo de componentes de Java.
- Commons Dormant: Un repositorio de componentes que se encuentran actualmente inactivas.

3.12.2. Hadoop Distributed File System (HDFS)

Es un sistema de archivos distribuido que proporciona acceso de alto rendimiento a datos. Fue creado a partir del Google File System (GFS). HDFS se encuentra optimizado para grandes flujos y trabajar con ficheros grandes en sus lecturas y escrituras. [24] La escalabilidad y disponibilidad son otras de sus claves, gracias a la replicación de los datos y tolerancia a los fallos.

3.12.2.1. Características

HDFS posee características muy útiles para el manejo de grandes volúmenes de datos:

- Es adecuado para el almacenamiento y procesamiento distribuido.
- Permite el manejo de grandes archivos que pueden pesar cientos de Mega-Bytes, Giga-Bytes, Tera-Bytes, Peta-Bytes, etc.
- Proporciona permisos de archivo y autenticación.
- Los servidores ayudan a los usuarios a comprobar fácilmente el estado del clúster.
- Hadoop proporciona una interfaz de comandos para interactuar con HDFS.
- Tolerancia a fallos: para poder tener siempre disponible los datos en caso de ser requeridos, utiliza la replicación de estos en distintos nodos.
- Los nodos pueden hablar entre ellos para reequilibrar datos, mover copias, y conservar una alta replicación.

3.12.2.2. Arquitectura

HDFS sigue una arquitectura maestro-esclavo y contiene los siguientes elementos:

- Namenode: Actúa como el servidor maestro y se encarga de la administración del espacio de nombres del sistema de archivos, regula el acceso a los ficheros por parte de los clientes, ejecuta las operaciones del sistema de archivos como el cambio de nombre, cierre y apertura de archivos y directorios y realiza el mantenimiento del sistema de archivos y la metadata asociada a todos estos. Regula el acceso a los ficheros por parte de los clientes. Mantiene en memoria

la metadata del sistema de ficheros y control de los bloques de fichero que tiene cada DataNode.

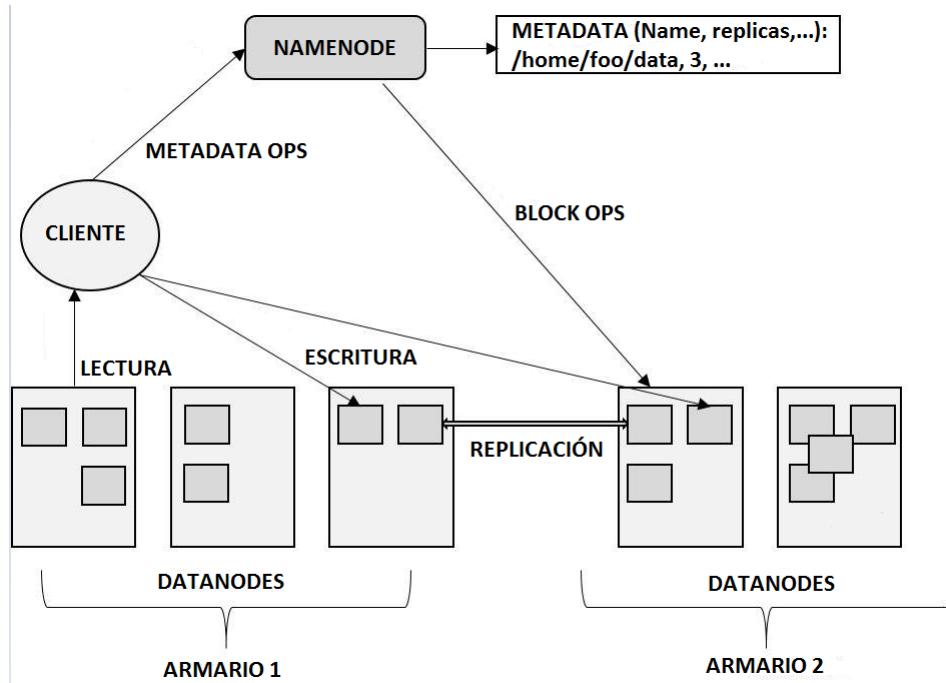


Figura 3.3: Arquitectura de HDFS.

- Datanode: Son los responsables de leer y escribir las peticiones de los clientes. Almacena y distribuye los bloques entre los distintos nodos. Permiten realizar operaciones tales como creación, supresión, con lo que la replicación de acuerdo con las instrucciones del namenode. Los ficheros están formados por bloques, estos se encuentran replicados en diferentes nodos. La distribución de los bloques es realizada cuando reciben un aviso desde un namenode o algún cliente (explicado en el siguiente punto) solicita una distribución. Una vez realizada la distribución, estos realizan un reporte al namenode, este reporte se realiza periódicamente y contiene los bloques los cuales están siendo almacenados en ese nodo en específico.
 - Bloque: En general los datos de usuario se almacenan en los archivos de HDFS. El archivo se divide en uno o más segmentos y son almacenados en los nodos. Estos segmentos, es decir, la cantidad mínima de datos que HDFS puede leer o escribir se llama un bloque. El tamaño de bloque por defecto es de 64 MB, pero puede ser modificado.
 - Clientes: son los usuarios del sistema de archivos

HDFS se gestiona a través de un servidor NameNode dedicado para alojar el índice de sistema de archivos y un NameNode secundario destinado a generar instantáneas de estructuras de memoria del NameNode principal. De esta manera, se evita la corrupción del sistema de archivos y la reducción de pérdida de datos.

3.12.3. Yet Another Resource Negotiator (YARN)

Apache Hadoop YARN (por las siglas en inglés de “otro negociador de recursos”) es una tecnología de administración de clústeres. Facilita la planificación de tareas y gestión de recursos de clúster. Nace para dar solución a una idea fundamental: Dividir las dos funciones principales del JobTracker (NameNode), es decir, tener en servicios o demonios totalmente separados e independientes la gestión de recursos por un lado y, por otro, la planificación y monitorización de las tareas o ejecuciones. [25]

Gracias a YARN, Hadoop tiene un entorno de gestión de recursos y aplicaciones distribuidas dónde se pueden implementar múltiples aplicaciones de procesamiento de datos totalmente personalizadas y específicas para realizar una tarea en cuestión. YARN combina un administrador central de recursos que reconcilia la forma en que las aplicaciones utilizan los recursos del sistema de Hadoop con los agentes de administración de nodo que monitorean las operaciones de procesamiento de nodos individuales del clúster. Ejecutándose en clústeres de hardware básicos, Hadoop ha atraído un interés particular como zona de espera y de almacenamiento de datos para grandes volúmenes de datos estructurados y no estructurados destinados al uso en aplicaciones de analítica. Separar HDFS de MapReduce con YARN hace al ambiente Hadoop más adecuado para las aplicaciones operativas que no pueden esperar para que terminen los trabajos por lotes.

3.12.3.1. Características

- Procesamiento: soporta distintos modelos de procesamiento y posee la capacidad de adaptarse a muchos más.
- Compatibilidad: las aplicaciones existentes de MapReduce pueden correr en YARN sin inconvenientes.
- Escalabilidad: YARN evita ciertos problemas de cuello de botella en grandes clúster.

3.12.3.2. Arquitectura

Los elementos que intervienen son:

- Resource Manager: asigna los recursos del clúster de manera abstracta a través de contenedores (Containers) los cuales incorporan elementos como memoria, CPU, disco, red, entre otros.
- Node Manager: hay uno por nodo esclavo, es el responsable de la monitorización y gestión de los recursos. Recoge las directrices del ResourceManager y crea contenedores en base a los requerimientos de la tarea.
- Application master: se despliega junto al NodeManager. Controla la monitorización y la ejecución de las tareas usando el contenedor. Puede ser alguna

librería específica de algún framework que es la encargada de negociar recursos con el ResourceManager y coordinar las tareas con el NodeManager. Proporciona tolerancia a fallos a nivel de tarea.

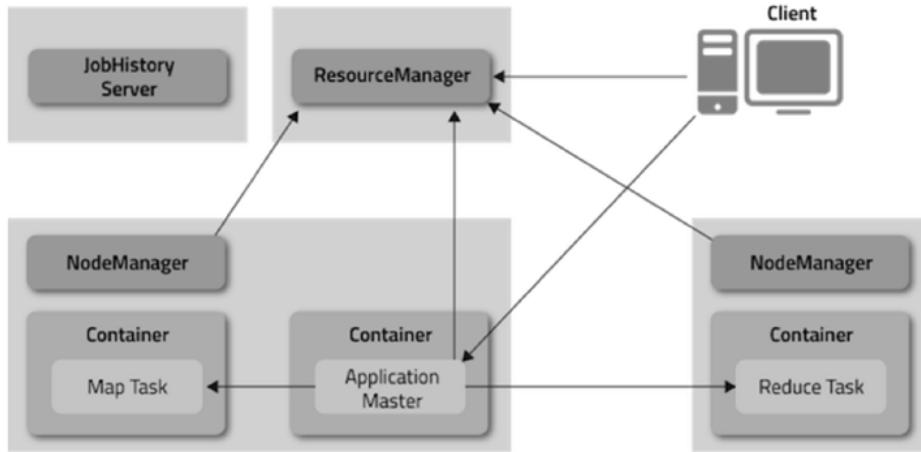


Figura 3.4: Arquitectura de YARN.

- Container: es la unidad básica de la asignación. El contenedor se define con atributos como la memoria, CPU, disco, entre otros, aplicaciones como procesamiento gráfico y MPI.
- History Server: mantiene la historia de todos los Jobs.

3.12.4. MapReduce

MapReduce fue desarrollado por Google y expuesto al resto del mundo en una publicación hecha por ellos mismos en diciembre del 2004, Google usa MapReduce para indexar páginas web. MapReduce es un modelo de programación con una implementación asociada al procesamiento y generación de grandes cantidades de datos. Los usuarios especifican una función de *Map* que procesa pares clave/valor para generar un grupo intermedio de pares clave/valor y una función de *Reduce* que combina todos los valores intermedios. [26]

Es un subproyecto del proyecto Hadoop. Posee una arquitectura maestro-esclavo y es un paradigma de programación que permite la escalabilidad masiva a través de cientos o miles de servidores en un clúster Hadoop. Cuenta con un servidor maestro o JobTracker y varios servidores esclavos o TaskTrackers, uno por cada nodo del clúster. El JobTracker es el punto de interacción entre los usuarios y el framework MapReduce. Los usuarios envían trabajos MapReduce al JobTracker, que los pone en una cola de trabajos pendientes y los ejecuta en el orden de llegada.

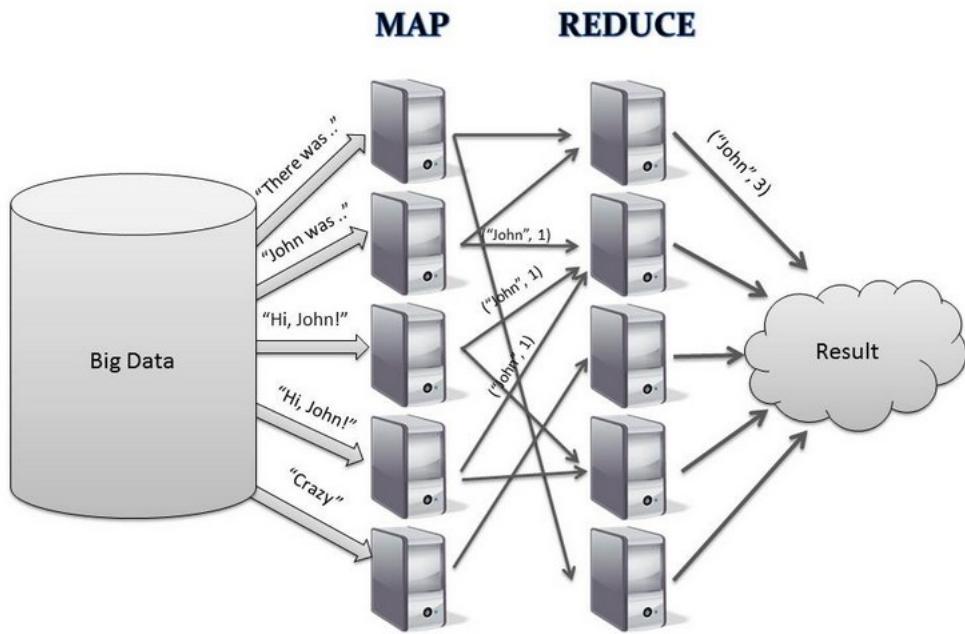


Figura 3.5: Ejemplo de MapReduce.

El JobTracker gestiona la asignación de tareas y delega las tareas a los TaskTrackers. Los TaskTrackers ejecutan tareas bajo la orden del JobTracker y también manejan el movimiento de datos entre la fase Map y Reduce.

3.12.4.1. JobTracker

- Capacidad para manejar metadatos de trabajos.
- Estado de la petición del trabajo.
- Estado de las tareas que se ejecutan en TaskTracker.
- Decide sobre la programación.
- Hay exactamente un JobTracker por cluster.
- Recibe peticiones de tareas enviadas por el cliente.
- Programa y monitoriza los trabajos MapReduce con TaskTrackers.

3.12.4.2. TaskTracker

- Ejecuta las solicitudes de trabajo de JobTrackers.
- Obtiene el código que se ejecutará.
- Aplica la configuración específica del trabajo.

- Comunicación con el JobTracker: Envíos de la salida, finalizar tareas, actualización de tareas, etc.

3.12.4.3. Map

La función Map recibe como parámetros un par clave-valor y devuelve una lista de pares. Esta función se encarga del mapeo y se aplica a cada elemento de la entrada de datos, por lo que se obtendrá una lista de pares por cada llamada a la función Map. Después se agrupan todos los pares con la misma clave de todas las listas, creando un grupo por cada una de las diferentes claves generadas. No hay requisito de que el tipo de datos para la entrada coincida con la salida y no es necesario que las claves de salida sean únicas.

3.12.4.4. Reduce

La función Reduce se aplica en paralelo para cada grupo creado por la función Map(). La función Reduce se llama una vez para cada clave única de la salida de la función Map. Junto con esta clave, se pasa una lista de todos los valores asociados con la clave para que pueda realizar alguna fusión y producir un conjunto más pequeño de los valores.

Cuando se inicia la tarea Reduce, la entrada se encuentra dispersa en varios archivos a través de los nodos en las tareas de Map. Los datos obtenidos de la fase Map se ordenan para que los pares clave-valor sean contiguos (fase de ordenación, sort fase), esto hace que la operación Reduce se simplifique ya que el archivo se lee secuencialmente.

Si se ejecuta el modo distribuido estos necesitan ser primero copiados al filesystem local en la fase de copia. Una vez que todos los datos están disponibles a nivel local se adjuntan a una fase de adición, el archivo se fusiona (merge) de forma ordenado. Al final, la salida consistirá en un archivo de salida por tarea reduce ejecutada.

Por lo tanto, N archivos de entrada generará M mapas de tareas para ser ejecutados y cada mapa de tareas generará tantos archivos de salida como tareas Reduce hayan configuradas en el sistema.

3.12.5. Ecosistema Hadoop

El ecosistema de Hadoop posee un conjunto de herramientas muy diverso, que crece día tras día, por lo que es difícil saber de todos los proyectos que interactúan con Hadoop de alguna forma, las cuales a su vez poseen subconjuntos, en este capítulo se mencionará algunos de ellos y una breve descripción.

3.12.5.1. Administración de los datos

Son herramientas que permiten el almacenamiento y el manejo de datos, como HDFS y YARN que fueron mencionados anteriormente.

3.12.5.2. Acceso a los datos

Son herramientas que se encargan del acceso a los datos por parte de los usuarios y permiten la lectura, escritura y procesamiento de los datos.

3.12.5.2.1. Apache Accumulo

Apache Accumulo es un store clave/valor distribuido, escalable y de alto rendimiento. Se basa en el diseño de Google BigTable y se construye sobre Hadoop, Zookeeper y Thrift. [27]

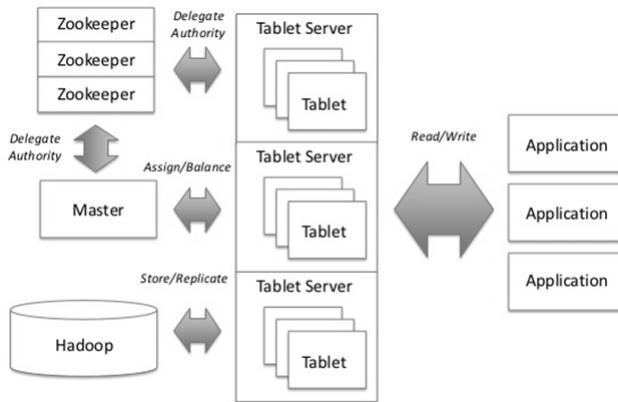


Figura 3.6: Arquitectura Apache Accumulo.

Accumulo permite el manejo de datos a nivel de celdas, lo cual es una funcionalidad muy importante debido a que se puede restringir el acceso a los datos a ciertos usuarios en específico. Permite también la mezcla de distintos datos los cuales pueden estar restringidos o no, las reglas que pueden ser aplicadas a los datos pueden llegar a ser muy específicas.

3.12.5.2.2. Apache Hbase

HBase, se trata de la base de datos de Hadoop. HBase es el componente de Hadoop a usar cuando se requiere escrituras/lecturas en tiempo real y acceso aleatorio para grandes conjuntos de datos. Es una base de datos distribuida orientada a columnas que funciona sobre HDFS, eso quiere decir que no sigue el esquema relacional. [28]

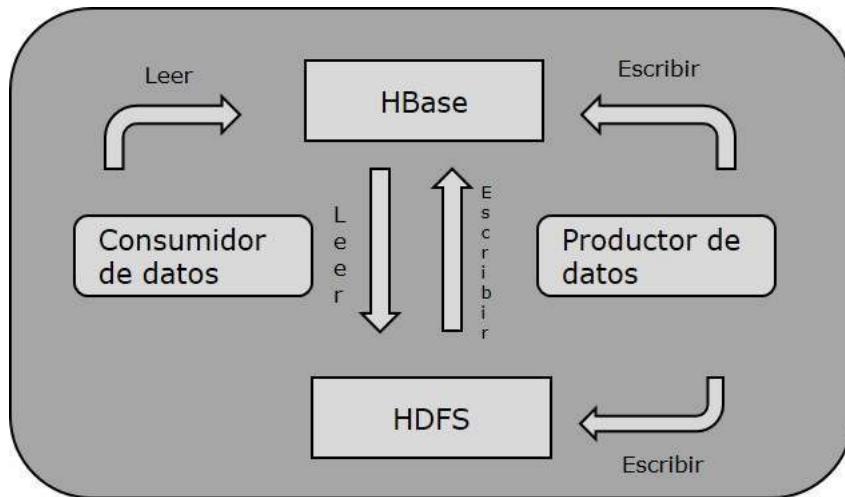


Figura 3.7: Modelo utilizando Apache HBase.

Características de HBase:

- Escalabilidad linear y modular.
- Soporte para caídas de nodos.
- Proporciona lectura coherente y escrituras.
- Se integra con Hadoop, tanto como un origen y un destino.
- Compatibilidad con trabajos MapReduce.
- Proporciona replicación de datos en clústeres.

Las aplicaciones de HBase:

- Apache HBase se utiliza para tener al azar y en tiempo real de acceso de lectura/escritura a los grandes datos.
- Alberga las tablas de gran tamaño en la parte superior de los grupos de hardware de productos básicos.
- Se usa cuando es necesario escribir aplicaciones pesadas.
- HBase se utiliza cada vez que se necesite proporcionar un rápido acceso aleatorio a los datos disponibles.
- Empresas como Facebook, Twitter, Yahoo y Adobe usan HBase internamente.

HDFS	HBase
HDFS es un sistema de ficheros distribuido adecuado para almacenar archivos de gran tamaño.	HBase es una base de datos creada en la parte superior de la HDFS.
HDFS no admite búsquedas rápidas registro individual.	HBase proporciona búsquedas rápidas tablas más grandes.
Proporciona una alta latencia procesamiento por lotes; un concepto de procesamiento por lotes.	Proporciona acceso de baja latencia a filas de miles de millones de registros (acceso aleatorio).
Sólo proporciona acceso secuencial de los datos.	HBase internamente usa tablas Hash y proporciona acceso aleatorio, y que almacena los datos en archivos indexados HDFS búsquedas más rápido.

Tabla 3.2: Comparación entre HBase y HDFS.

3.12.5.2.3. Apache Hive

Hive es un sistema de Data Warehouse para Hadoop que facilita el uso de la agregación de los datos, ad-hoc queries, y el análisis de grandes datasets almacenados en Hadoop. Hive proporciona métodos de consulta de los datos usando un lenguaje parecido al SQL, llamado HiveQL. Posee interfaces JDBC/ODBC, por lo que empieza a funcionar su integración con herramientas de inteligencia de negocios. [29]

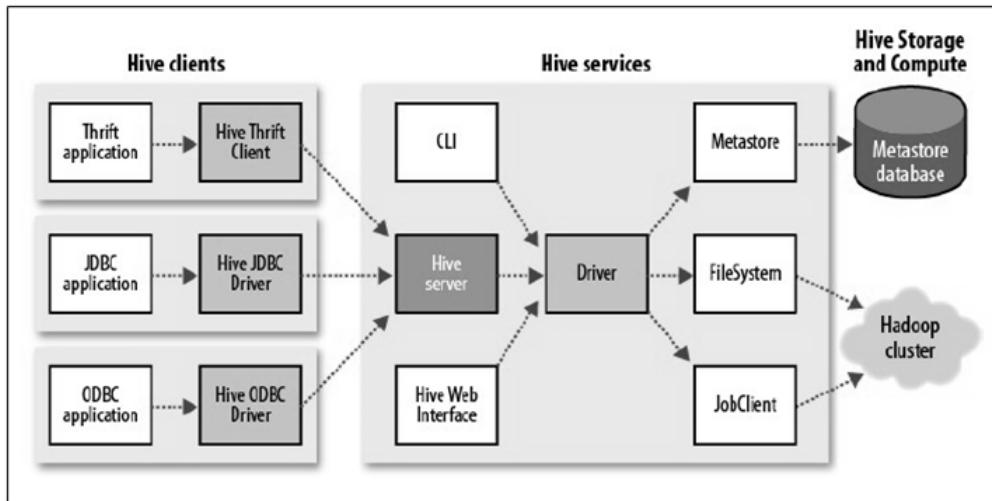


Figura 3.8: Arquitectura Apache Hive.

Características de Hive:

- Esquema que almacena en una base de datos y se procesan los datos en HDFS.
- Está diseñado para OLAP.
- Proporciona un lenguaje de consulta parecido a SQL, HiveQL (HQL).
- Es familiar, rápido, escalable y extensible.

3.12.5.2.4. Apache Storm

Apache Storm es un sistema que sirve para recuperar streams de datos en tiempo real desde múltiples fuentes de manera distribuida, tolerante a fallos y en alta disponibilidad. Storm está principalmente pensado para trabajar con datos que deben ser analizados en tiempo real, por ejemplo datos de sensores que se emiten con una alta frecuencia o datos que provengan de las redes sociales donde a veces es importante saber qué se está compartiendo en este momento. [30]

Se compone de dos partes principalmente. La primera es la que se denomina Spout y es la encargada de recoger el flujo de datos de entrada. La segunda se denomina Bolt y es la encargada del procesado o transformación de los datos.

3.12.5.2.5. Apache Mahout

Es una librería de los algoritmos más famosos de aprendizaje automático (Machine Learning), fue implementado sobre Hadoop utilizando el paradigma de MapReduce. Con los datos almacenados en HDFS, Mahout provee las herramientas necesarias para poder aplicar algoritmos sobre estos.

Proporciona bibliotecas de Java para las operaciones matemáticas comunes (centrado en álgebra lineal y estadística) y las colecciones de Java primitivos. También ha añadido un número de algoritmos matemáticos de bajo nivel que los usuarios pueden encontrar útil. [31]

Mahout ayuda a descubrir patrones en grandes datasets. Tiene algoritmos de recomendación, clustering y clasificación. Algunos de los algoritmos que provee Mahout son los siguientes:

Algoritmo	Descripción breve	Caso de uso
Regresión logística, resuelta por gradiente estocástico descendiente (SGD).	Clasificador brillante, rápido, simple y secuencial, capaz de aprendizaje online en entornos exigentes.	Recomienda publicidad a los usuarios, clasifique texto en categorías.
Modelos oculitos de Markov (HMM).	Implementaciones secuenciales y paralelas del algoritmo clásico de clasificación diseñado para modelar procesos del mundo real cuando el proceso de generación subyacente es desconocido.	Etiquetado de texto parte-del-discurso; reconocimiento del discurso.
Descomposición de valor singular (SVD).	Diseñado para reducir el ruido en matrices grandes, haciendo con esto que sean más pequeñas y que sea más fácil trabajar con ellas.	Como precursor del almacenamiento en clúster, los recomendadores y la clasificación para realizar selección de recursos automáticamente.
Almacenamiento en clúster Dirichlet.	Enfoque de almacenamiento en clúster basado en modelo, que determina la propiedad con base en si los datos se ajustan al modelo subyacente.	Útil cuando los datos tienen sobreposición o jerarquía.
Almacenamiento en clúster espectral.	Es una familia de enfoques similares que usa un enfoque basado en gráficas para determinar la membresía a clúster.	Como todos los algoritmos de almacenamiento en clúster, es útil para explorar conjuntos de datos grandes y no vistos.
Almacenamiento en clúster Minhash.	Utiliza una estrategia de hash para agrupar elementos similares, produciendo así clústeres.	Igual a otros enfoques de clúster.
Numerosas mejoras de recomendador.	Co-ocurrencia distribuida, SVD, mínimos cuadrados alternantes.	Sitios de citas, e-commerce, recomendaciones de películas o de libros.
Colocaciones.	Implementación de colocación reducida por correlacionamiento.	Encontrando frases estadísticamente interesantes en texto.

Tabla 3.3: Algoritmos Mahout.

3.12.5.3. Integración

Aquellas herramientas que facilitan la extracción, transformación, replicación, entre otros, de los datos, son las herramientas de integración.

3.12.5.3.1. Apache Falcon

Falcon es un sistema de procesamiento y manejo de carga de datos destinada a facilitar a los consumidores finales a bordo de sus procesamiento de datos. Establece relación entre los diversos elementos de procesamiento de datos y en un entorno de Hadoop. Posee soporte para el manejo de datos finales, integración con MetaStore/Hive/HCatalog. Proporciona notificaciones al cliente final sobre la base de la disponibilidad de los grupos de carga de datos.

Básicamente, es una herramienta de software que simplifica la creación y el manejo de tuberías (pipelines) de procesamiento de datos. [32]

3.12.5.3.2. Apache Flume

Apache Flume es un producto que forma parte del ecosistema Hadoop, y conforma una solución Java distribuida y de alta disponibilidad para recolectar, agregar y mover grandes cantidades de datos desde diferentes fuentes a un data store centralizado.

Surge para subir datos de aplicaciones al HDFS de Hadoop. Su Arquitectura se basa en flujos de streaming de datos, ofrece mecanismos para asegurar la entrega y mecanismos de failover y recuperación. Ofrece una gestión centralizada. [33]

3.12.5.3.3. Apache Sqoop

Apache Sqoop (“Sql-to-Hadoop”), es una herramienta diseñada para transferir de forma eficiente bulk data entre Hadoop y sistemas de almacenamiento con datos estructurados, como bases de datos relacionales [34]. Algunas de sus características son:

- Permite importar tablas individuales o bases de datos enteras a HDFS.
- Genera clases Java que permiten interactuar con los datos importados.
- Además, permite importar de las bases de datos SQL a Hive.

3.12.5.4. Operaciones

Son las herramientas que permiten monitorear, administrar y realizar operaciones sobre un clúster Hadoop.

3.12.5.4.1. Apache Ambari

Es un proyecto que pertenece a la distribución Hortonworks, que facilita la gestión de Hadoop. Ofrece una interfaz web intuitiva y fácil de usar para la gestión de Hadoop y además proporciona una API REST.[35] Ambari permite a los administradores del sistema:

- Monitorizar el clúster Hadoop.
- Ofrece un panel de control para vigilancia de la salud y el estado del cluster Hadoop.
- Se encarga de la instalación de los paquetes de Hadoop en el clúster.
- Un asistente paso a paso para la instalación de servicios de Hadoop a través de múltiples equipos.
- Proporciona la forma de gestionar gestión central para iniciar, detener y volver a configurar los servicios de Hadoop en todo el clúster.

3.12.5.4.2. Apache Zookeeper

Zookeeper es un proyecto de Apache que proporciona una infraestructura centralizada y de servicios que permiten la sincronización del cluster. ZooKeeper mantiene objetos comunes que se necesiten en grandes entornos de cluster. [36]

3.12.5.4.3. Apache Oozie

Apache Oozie es un sistema de programación de flujo de trabajo basado en servidor para administrar los trabajos de Hadoop. Los flujos de trabajo en Oozie se definen como una colección de flujos de control y nodos de acción en un grafo dirigido acíclico.

Los nodos de flujo de control definen el comienzo y el final de un flujo de trabajo (inicio, final y los nodos de fallo), así como un mecanismo para controlar la ruta de ejecución del flujo de trabajo. Los nodos de acción son el mecanismo por el cual un flujo de trabajo provoca la ejecución de una tarea de cálculo/procesamiento.

Oozie se implementa como una aplicación web en Java que se ejecuta en un contenedor de servlets Java y se distribuye bajo la licencia Apache 2.0. [37]

3.12.5.5. Seguridad

Apache Hadoop provee herramientas de monitoreo y administración que aportan seguridad a los datos, mediante autorización, autenticación, protección, auditoría, entre otros.

3.12.5.5.1. Apache Knox

El Apache Knox Gateway es una puerta de enlace API REST para interactuar con los racimos de Hadoop. El Knox Gateway proporciona un único punto de acceso para todas las interacciones REST con racimos de Hadoop. En esta capacidad, Knox es capaz de proporcionar la funcionalidad valiosa de ayudar en el control, la integración, el seguimiento y la automatización de las necesidades administrativas y analíticas que son críticas en la empresa. [38]

- Autenticación (LDAP y proveedor de autenticación de Active Directory).
- Federación/SSO (encabezado HTTP basado federación de identidades).
- Autorización (Service Level Autorización).
- Revisión de cuentas.

3.12.5.5.2. Apache Ranger

Ranger es un framework que permite supervisar y gestionar la seguridad de datos completa a través de la plataforma Hadoop.

La visión con Ranger es proporcionar seguridad integral en todo el ecosistema Hadoop. Con la llegada de Apache Yarn, la plataforma Hadoop ahora puede soportar una verdadera arquitectura de conjunto de datos. La seguridad de los datos dentro de Hadoop necesita evolucionar para soportar múltiples casos de uso para el acceso de datos, mientras que también proporciona un marco para la administración central de las políticas y el seguimiento de acceso de los usuarios de seguridad. [39]

3.12.6. Apache Spark

Apache Spark es una alternativa para el procesamiento de grandes volúmenes de datos (Big Data) que ha suplantado a muchas herramientas. Es una plataforma de computación de código abierto para análisis y procesos avanzados, que tiene muchas ventajas sobre Hadoop. Desde el principio, Spark fue diseñado para soportar en memoria algoritmos iterativos que se pudiesen desarrollar sin escribir un conjunto de resultados cada vez que se procesaba un dato. Esta habilidad para mantener todo en memoria es una técnica de computación de alto rendimiento aplicado al análisis avanzado, la cual permite que Spark tenga unas velocidades de procesamiento que sean 100 veces más rápidas que las conseguidas utilizando MapReduce.

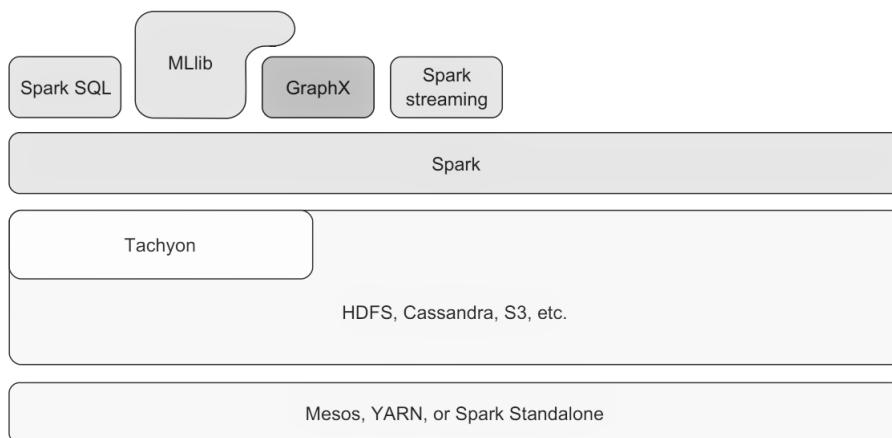


Figura 3.9: Apache Spark.

Spark posee un framework integrado para implementar análisis avanzados que incluye la librería MLlib, GraphX, Spark Streaming, y la herramienta de consulta Shark. Esta plataforma asegura a los usuarios la consistencia en los resultados a través de distintos tipos de análisis. [40]

Al igual que Hadoop, Spark provee el API MapReduce, la diferencia es que Spark almacena los datos en memoria RAM del clúster y Hadoop en el disco duro. También proporciona procesamiento de datos en paralelo lo que lo hace ideal para los problemas de Big Data en el mundo real, ya que a menudo se requiere tanto procesamiento de grafos en paralelo como de datos. Spark mantiene la escalabilidad lineal y la tolerancia a fallos de MapReduce, pero amplía sus bondades gracias a varias funcionalidades: DAG y RDD.

3.12.6.1. Spark SQL

Spark SQL trae un soporte nativo de SQL para Spark y agiliza el proceso de consulta de datos almacenados tanto en RDD como en otras fuentes. [40] Difumina convenientemente las líneas entre RDD y las tablas relacionales. La unificación de estas potentes abstracciones hace que sea fácil para los desarrolladores entremezclar comandos de consulta SQL sobre datos externos para análisis complejos, todo dentro de una sola aplicación. Concretamente, permite a los desarrolladores:

- Importar datos relacionales a partir de archivos de Parquet y tablas de Hive.
- Ejecutar consultas SQL sobre los datos importados y RDD existentes.
- Escribir fácilmente sobre RDD fuera de Hive o archivos Parquet.

3.12.6.2. MLlib

MLlib es un subproyecto de Spark que provee primitivas de aprendizaje automático escalable y sencillo.[40] Esta compuesto por los algoritmos más comunes,

como los de clasificación, clusterización, regresión, filtrado colaborativo, reducción de dimensionalidad, entre otros. Se encuentra dividido en 2 paquetes:

- spark.mllib contiene el API original realizado sobre los RDDs.
- spark.ml provee un API de alto nivel realizado sobre los DataFrames para construir ML pipelines.

Usualmente se utiliza spark.ml debido a que este API es más flexible y versátil manejando DataFrames.

3.12.6.3. Spark streaming

Spark Streaming provee un API con un lenguaje integrado para procesos de streaming, que permiten desarrollar operaciones de streaming al mismo tiempo que los lotes de trabajo. Soporta Java, Scala y Python.[40]

Puede procesar datos utilizando una variedad de algoritmos y funciones tales como map, reduce, join y window, y puede extraer datos de una gran variedad de fuentes, incluyendo flujos provenientes de Apache Kafka, Apache Flume, Amazon Kinesis y Twitter, así como de sensores y dispositivos conectados por medio de sockets TCP. También se pueden procesar datos almacenados en sistemas de archivos como HDFS o Amazon S3.

En otras palabras Spark Streaming lo que hace es tomar un flujo de datos continuo y convertirlo en un flujo discreto llamado DStream, formado por paquetes de datos. Internamente, lo que sucede es que Spark Streaming almacena y procesa estos datos como una secuencia de RDDs.

3.12.6.4. Grafo Acíclico Dirigido (Directed Acyclic Graph (DAG))

Es un grafo dirigido que no tiene ciclos, es decir, para cada nodo del grafo no hay un camino directo que comience y finalice en dicho nodo. Un vértice se conecta a otro, pero nunca a si mismo. [41]

Spark soporta el flujo de datos acíclico. Cada tarea de Spark crea un DAG de etapas de trabajo para que se ejecuten en un determinado cluster. En comparación con MapReduce, el cual crea un DAG con dos estados predefinidos (Map y Reduce), los grafos DAG creados por Spark pueden tener cualquier número de etapas. Spark con DAG es más rápido que MapReduce por el hecho de que no tiene que escribir en disco los resultados obtenidos en las etapas intermedias del grafo. MapReduce, sin embargo, debe escribir en disco los resultados entre las etapas Map y Reduce. Gracias a una completa API, es posible programar complejos hilos de ejecución paralelos en unas pocas líneas de código.

3.12.6.5. Conjuntos Distribuidos Resilientes (Resilient Distributed Dataset (RDD))

Conjuntos Distribuidos Resilientes tiene un rol importante en Spark para la representación de grafos. Consiste en subconjuntos de datos distribuidos llamados particiones que debe ser cargados en la memoria de los clusters, con la finalidad de que los datos puedan ser vistos como arreglos de memoria.

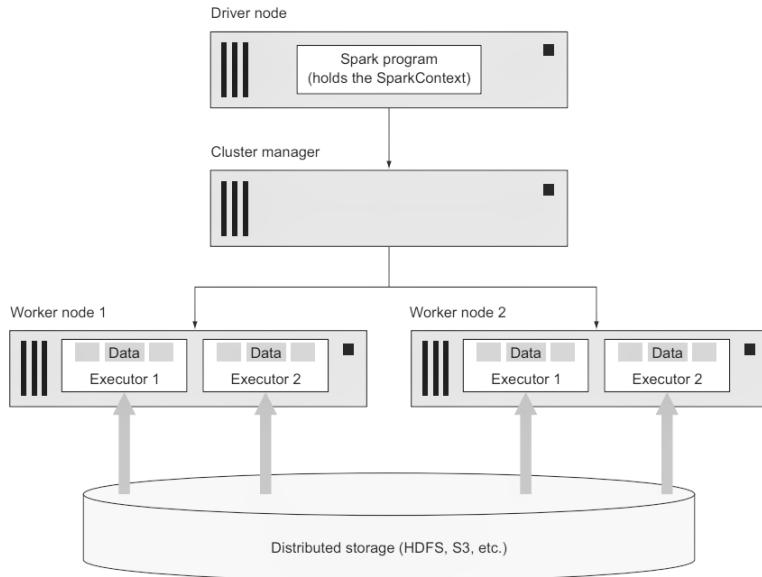


Figura 3.10: Conjuntos Distribuidos Resilientes.

Apache Spark mejora con respecto a los demás sistemas en cuanto a la computación en memoria. RDD permite a los programadores realizar operaciones sobre grandes cantidades de datos en clusters de una manera rápida y tolerante a fallos. Surge debido a que las herramientas existentes tienen problemas que hacen que se manejen los datos inefficientemente a la hora de ejecutar algoritmos iterativos y procesos de minería de datos. En ambos casos, mantener los datos en memoria puede mejorar el rendimiento considerablemente. [42]

Una vez que los datos han sido leídos como objetos RDD en Spark, pueden realizarse diversas operaciones mediante sus APIs. Los dos tipos de operaciones que se pueden realizar son:

- Transformaciones: tras aplicar una transformación, obtenemos un nuevo y modificado RDD basado en el original.
- Acciones: una acción consiste simplemente en aplicar una operación sobre un RDD y obtener un valor como resultado, que dependerá del tipo de operación.

Dado que las tareas de Spark pueden necesitar realizar diversas acciones o transformaciones sobre un conjunto de datos en particular, es altamente recomendable y

beneficioso en cuanto a eficiencia el almacenar RDDs en memoria para un rápido acceso a los mismos. Mediante la función cache() se almacenan los datos en memoria para que no sea necesario acceder a ellos en disco.

El almacenamiento de los datos en memoria caché hace que los algoritmos ejecutados de aprendizaje automático, que realizan varias iteraciones sobre el conjunto de datos de entrenamiento sean más eficientes. Además, se pueden almacenar versiones transformadas de dichos datos.

3.12.6.6. Ventajas de Spark

- Spark tiene más potencia que hadoop:

Para empezar, Spark es un framework de análisis distribuido en memoria y permite ir más allá de las operaciones en batch de Hadoop MapReduce: procesamiento de streaming, machine learning (MLlib), cálculo de grafos (GraphX), integración con lenguaje R (Spark R) y análisis interactivos.

Con todo esto, ahora se puede desarrollar nuevos proyectos de big data con menos presupuesto y soluciones más completas.

- Spark es rápido, muy rápido:

Spark puede ejecutar análisis de varios órdenes de magnitud más rápido que los despliegues de Hadoop existentes. Esto significa una mayor interactividad, la experimentación más rápido y mayor productividad para los analistas.

- Spark puede coexistir con una arquitectura Big Data:

Puede coexistir con las instalaciones existentes de Hadoop y añadir nuevas funcionalidades. Spark se integra perfectamente con Hadoop y en muchos de los proyectos se utiliza/almacenan los datos que están en el sistema de fichero de Hadoop HDFS y/o se ejecutan los procesos de Spark usando YARN de Hadoop 2.0. Además puede funcionar con muchos otros productos de Big Data como: CassandraDB, Google Big Query, almacenamiento de Amazon S3, Elastic Search, etc.

- Spark entiende SQL:

El módulo Spark Sql es capaz de usar fuentes de datos existentes (HIVE, CassandraDB, MongoDB, JDBC, etc), se puede usar para gestionar las fuentes internas de datos (RDDs y/o DataFrames) como si fueran tablas estructuradas. Aunque Spark SQL no es la implementación más robusta y completa del mercado ya está lista para ser usada.

- Spark mima a los desarrolladores:

Cuando una tecnología encanta a los desarrolladores se convierten en early adopters y empiezan a usarla y disfrutarla. Spark es un ejemplo de esto, cuando usan Spark solo tiene que dedicarse a resolver el problema. Spark se ha

programado con el lenguaje Scala que un nuevo lenguaje funcional y orientado a objetos. Gracias a Scala son capaces de programar de manera muy concisa y fluida soluciones que antes requerían cientos de líneas. Además se puede programar en Python, R e incluso en Java.

- Spark empieza a ser el motor de Big Data:

Ahora mismo Apache Spark forma muchos proyectos de Big Data y empresas como IBM, Microsoft, Amazon, Google lo integran con sus productos de Big Data.

Apache Spark es similar a Apache Hadoop en el sentido que almacenan los datos de forma distribuida utilizando clusters, la diferencia es que Apache Spark los almacena en memoria (RAM) y Hadoop en disco. Hadoop tiene 2 inconvenientes, procesar consultas interactivas y algoritmos iterativos, por lo tanto nace Apache Spark para solventar estos problemas.



Figura 3.11: Apache Hadoop vs Apache Spark

3.12.7. Distribuciones Hadoop

3.12.7.1. Cloudera

Cloudera fue fundada en 2008 por algunas de las mentes más brillantes de las empresas líderes de Silicon Valley, incluyendo Google (Christophe Bisciglia), Yahoo (Amr Awadallah), Oracle (Mike Olson) y Facebook (Jeff Hammerbacher). Se fundó con el objetivo de ayudar a las empresas a usar Hadoop para obtener más valor de todos sus datos. Ofrece una plataforma de datos basada en Hadoop, rápida, fácil de usar y segura. Basada al 100 % en software de código abierto y estándares abiertos, la plataforma de Cloudera proporciona una mayor flexibilidad, más control de los

costes y unos mejores resultados para la empresa. [43]

Cloudera Enterprise ofrece 3 opciones para utilizar Apache Hadoop, local, en una infraestructura basada en un entorno local o en la nube:

- QuickStart VM o Docker Image: Permite la utilización de Cloudera mediante la utilización de máquinas virtuales y contenedores Docker. Provee un entorno de pruebas (Sandbox) con diversos ejemplos y herramientas.
- Cloudera Manager: Provee una interfaz unificada para gestionar centros de datos empresariales.
- Cloudera Director: Provee un entorno en la nube que facilita la implementación y administración de Cloudera.

3.12.7.2. Hortonworks

Hortonworks es una compañía de software empresarial con sede en Santa Clara, California. La compañía se centra en el desarrollo y soporte de Apache Hadoop, posee un conjunto de procesos y herramientas que permiten el procesamiento distribuido de grandes conjuntos de datos a través de clusters. [44]

Hortonworks se formó en junio de 2011, financiado por Yahoo. Utiliza Apache Ambari como administrador del sistema y HDFS como sistema de archivos, algunas herramientas desarrolladas por Hortonworks son:

- Slider: framework el cual permite la gestión de recursos en tiempo real, según los requerimientos de las herramientas ejecutadas. Permite administrar de una mejor manera el clúster debido a que permite asignar mayor o menor cantidad de recursos según los requerimientos de las herramientas, esto anteriormente no era posible dependiendo solo de YARN.
- Tez: framework diseñado para trabajar sobre YARN, permite la manipulación de datos producidos en lotes.
- Phoenix: provee una interfaz SQL sobre HBase para así poder tratar este almacén de datos con un lenguaje muy parecido a SQL. Este realiza una optimización sobre distintas consultas a realizar lo cual provee un mayor rendimiento.
- YARN: administrador de recursos de Hadoop.

3.12.7.3. MapR

MapR es una empresa de software con sede en San José, California, que ofrece soluciones sobre el ecosistema Hadoop. La empresa contribuye a proyectos Apache Hadoop como HBase, Pig, Apache Hive, y Apache ZooKeeper. [45]

MapR es una empresa privada con financiación original de 9 millones de dólares por Lightspeed Venture Partners y New Enterprise Associates en el 2009. Utiliza de un sistema de archivos, una base de datos y un sistema de administración del clúster creados especialmente para esta distribución cuyos nombres son: MapR-FS, MapR-DB y MapR-Control System respectivamente.

- MapR-Control System. Es un sistema para poder controlar el ecosistema de la distribución, es equiparable a Apache Ambari. Provee sistemas de alarmas, una serie de medidores para observar el estado del clúster de una forma rápida. Este también funciona mediante una interfaz web que se comunica mediante un API el cual realiza todas las funcionalidades de esta herramienta.
- MapR-DB: Es una base de datos de calidad empresarial, fue creada basándose en el enfoque NoSQL. Es totalmente compatible con el ecosistema Hadoop, incluye soporte para distintas características como: modelo de datos basado en grandes columnas (al estilo Apache HBase), escalabilidad al estilo de Hadoop, localidad de datos con trabajos MapReduce, alta consistencia de datos, transacciones ACID a nivel de filas, entre otras.
- MapR-FS: Es un sistema de archivos distribuido basado en el funcionamiento de HDFS pero mejorando ciertos puntos de fallo que este tiene. A diferencia de HDFS, MapR-FS no es estructurado, no depende de una arquitectura en específico, este diseño busca solventar la problemática del punto de fallo simple que tiene HDFS relacionado con el NameNode.

3.12.7.4. Cloudera vs Hortonworks

Cloudera, así como Hortonworks están ambos construidos sobre el mismo núcleo de Apache Hadoop. Como tales, tienen más similitudes que diferencias. Ambos ofrecen distribuciones de Hadoop listas para la empresa. Las distribuciones han superado la prueba del tiempo generadas por los consumidores, y garantizan seguridad y estabilidad.

Además, proporcionan entrenamientos y servicios pagos para familiarizar a los nuevos usuarios que pisan el camino de grandes volúmenes de datos y análisis. Ambos poseen una arquitectura maestro-esclavo, soportan YARN y MapReduce, y han establecido comunidades que participan de manera activa.

Cloudera ha anunciado que su objetivo a largo plazo es convertirse en una empresa especializada en data hub (colección de datos procedentes de distintas fuentes, organizados para ser posteriormente distribuidos y compartidos), disminuyendo así la necesidad de utilizar almacenes de datos. Hortonworks, por el contrario, mantiene su firme posición como proveedor de una distribución de Hadoop, y se ha asociado con la compañía de almacenes de datos Teradata.

Mientras Cloudera CDH se puede ejecutar en un servidor Windows, HDP está disponible como un componente nativo de Windows. Un clúster Hadoop basado en

Windows puede utilizar Windows Azure a través del servicio HDInsight. Cloudera tiene un software de gestión propio llamado Cloudera Manager, una interfaz para consultas SQL (Impala), y Cloudera Search que permite un acceso fácil y en tiempo real de los productos. Hortonworks no tiene software propietario, utiliza Ambari para la gestión y Stinger para el manejo de consultas, y Apache Solr para las búsquedas de datos. Cloudera tiene uso comercial, mientras que Hortonworks tiene licencia de código abierto.

Cloudera también permite el uso de sus proyectos de forma gratuita, pero no incluye Cloudera Manager o cualquier otro software propietario. Cloudera tiene una versión de prueba de 60 días, Hortonworks es completamente libre. Cloudera ha sido el jugador de más edad en el mercado, con más de 350 clientes. Pero Hortonworks está igualando rápidamente y ha hecho más innovaciones en el ecosistema Hadoop en el pasado reciente.

3.13. Grafo

Un grafo es básicamente un conjunto no vacío (al menos contiene un elemento) de puntos llamados vértices y un conjunto de líneas llamadas aristas cada una de las cuales une dos vértices. Se llama lazo a una arista que une un vértice consigo mismo. Se dice que dos vértices son adyacentes si existe una arista que los une.[46]

Se dice que un grafo es simple si para cualesquiera dos vértices existe a lo sumo una arista que los une. En otro caso se denomina multigrafo. Si v es un vértice de un grafo, se denomina grado de v al número de aristas que inciden en el mismo (por convenio de considera que un lazo cuenta dos veces al determinar el grado de su vértice).

Algunas aplicaciones que pueden implementarse utilizando grafos son:

- Encontrar la ruta más corta en una aplicación de geo-mapping.
- Recomendación de productos, servicios o contactos personales.
- Deteminar los trabajos académicos más influyentes.

Algunos ejemplos que pueden ser representado como grafos, son los siguientes:

- Redes tecnológicas (technological networks).
- Redes biológicas (biological networks).
- Redes sociales (social networks).
- Redes de información (information networks).
- Redes de idiomas (language networks).
- Redes de información humana (human information network).
- Ciudades inteligentes (smart cities).

3.13.1. Redes tecnológicas (technological networks)

Son las redes diseñadas para la distribución de electricidad (energía), agua, gas, las redes de transportes (carreteras, ferrocarril, rutas aéreas), las redes telefónicas e internet (sólo las redes físicas de cables y postes, puesto que las redes de llamadas formarían parte de las denominadas redes sociales).

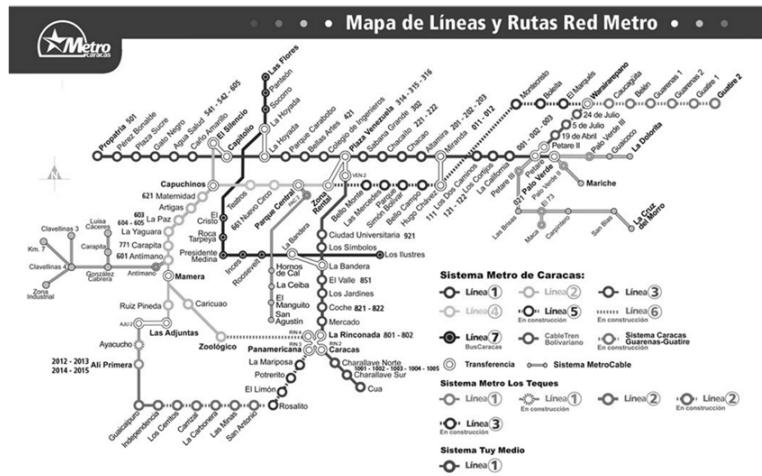


Figura 3.12: Red tecnológica. Red de metro.

3.13.2. Redes de información (information networks)

También denominadas redes de conocimiento. El ejemplo clásico de redes reales de esta categoría son las de citas o referencias de trabajos científicos. Otro ejemplo, ampliamente estudiado de redes de información es la World Wide Web, que es una red que contiene páginas informativas que se enlazan a través de hipervínculos. Al igual que las redes de citas, en la www también influyen aspectos sociales que trascienden el mero interés informativo de los vínculos.

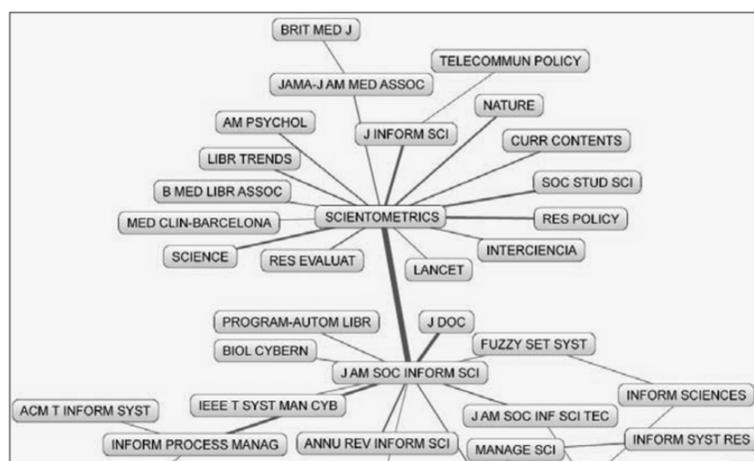


Figura 3.13: Red de información. Cocitación de revistas.

3.13.3. Redes sociales (social networks)

Las redes sociales están compuestas por individuos o grupos de individuos con patrones de contactos o interacciones entre ellos. Ejemplos de este tipo de redes son las relaciones de amistad, de negocios entre directivos de empresas, o entre familias a partir de sus matrimonios y descendencia (genealogías). Al análisis de este tipo de redes se asocian a menudo dificultades de imprecisión y subjetividad, debidas al reducido tamaño de las muestras que emplean y a los métodos utilizados para la recogida de datos: generalmente encuestas, cuestionarios o entrevistas. Para superar estas dificultades los investigadores han probado nuevos métodos de investigación en busca de muestras más numerosas y fiables mediante la utilización de grandes bases de datos.

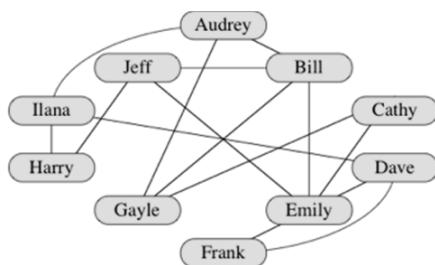


Figura 3.14: Red social. Colaboración científica.

Una Red Social es una estructura social integrada por personas, organizaciones o entidades que se encuentran conectadas entre sí por una o varios tipos de relaciones como ser: relaciones de amistad, parentesco, económicas, relaciones sexuales, intereses comunes, experimentación de las mismas creencias, entre otras posibilidades. Las Redes Sociales son un ejemplo claro de grandes volúmenes de datos, y realizar distintos tipos de análisis sobre estas puede ser muy útil. Muchas organizaciones se han dado cuenta que la gestión y un análisis completo de los datos, puede influir positivamente en la empresa si se realiza de forma adecuada y se toman las decisiones correctas. Big Data hace posible el análisis sobre la gran cantidad de datos que generan las redes sociales.

3.13.4. Redes biológicas (biological networks)

Los expertos en biomedicina quieren realizar descubrimientos científicos, un caso es, descubrir relaciones desconocidas . Por ejemplo, tomar dos enfermedades, enfermedades muy diferentes, cáncer colorrectal y la enfermedad de Alzheimer, y puede que tengan algún tipo de relación. Otros ejemplos son, las relaciones entre genes y proteínas, interacciones entre genes, señalización celular y asociaciones entre las enfermedades.

Son diversos los sistemas biológicos susceptibles de representarse en forma de redes. Las redes de reacciones metabólicas, las redes genéticas, los ecosistemas y cadenas tróficas, las redes neuronales o las vasculares son algunos de los ejemplos

de redes biológicas analizadas desde la perspectiva de la teoría de redes. Las redes alimentarias, por ejemplo, pueden ser descritas como un grafo con un conjunto finito de nodos (especies) y un conjunto finito de enlaces que asocian cada uno de esos nodos entre sí.

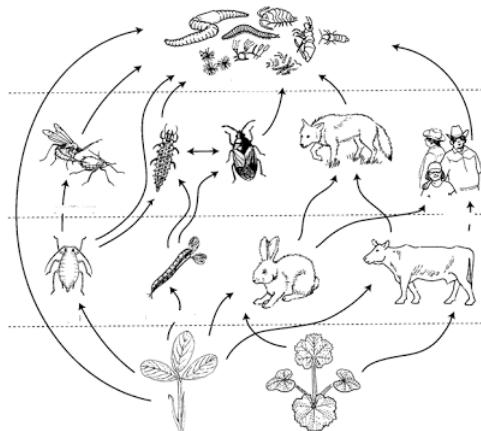


Figura 3.15: Red biológica. Cadena trófica.

3.13.5. Ciudades inteligentes (smart cities)

Una ciudad es un espacio geográficamente limitado, y contiene muchas redes diferentes que operan dentro del mismo dominio espacial. Cuenta con las redes de transporte, agua, cloacas, transmisión de energía, entre otros. Algunas de estas redes tienen múltiples subtipos, por ejemplo, las redes de transporte incluyen las redes de autobuses, metro, carreteras, ferroviaria, y así sucesivamente. Una ciudad inteligente se refiere a un tipo de desarrollo urbano basado en la sostenibilidad que es capaz de responder adecuadamente a las necesidades básicas de instituciones, empresas, y de los propios habitantes, tanto en el plano económico, como en los aspectos operativos, sociales y ambientales. Estas redes forman una infraestructura física y por lo tanto pueden ser representados mediante grafos, donde cada nodo tiene una coordenada geográfica.

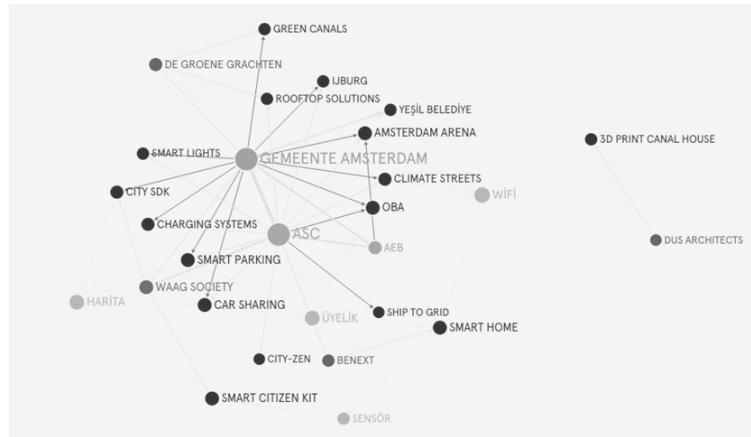


Figura 3.16: Ciudades inteligentes.

3.13.6. Tipos de datos que representan grafos

Existen distintos tipos de datos que pueden representar un grafo, algunos de estos son:

- Red (network).
- Árbol (tree).
- RDBMS o similar.
- Matriz dispersa (Sparse matrix).

Una red puede ser las rutas de las autopistas y las redes sociales. Un árbol no puede tener ciclos (loops). Cualquier RDBMS puede convertirse en un formato de grafo. Todo grafo puede asociarse como una matriz de adyacencia.

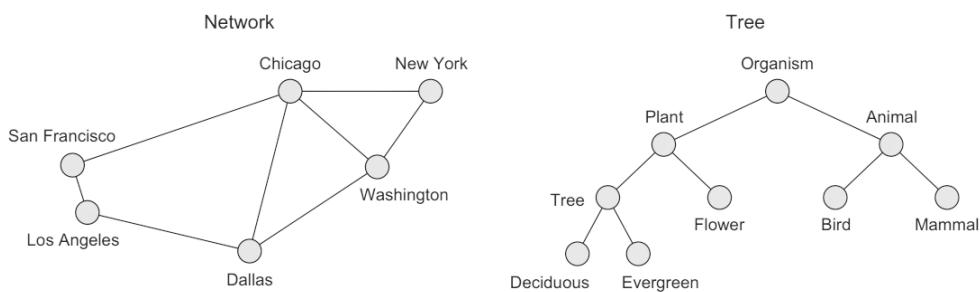


Figura 3.17: Red y árbol.

Usar el poderoso concepto de matrices de adyacencia tiene una implicación importante: un grafo es una estructura de datos alternativa y no es algo mágico. Algunos algoritmos, que de otra manera tendrían que lidiar con matrices difíciles de manejar, se puede hacer uso de la representación más comprimida de un grafo, especialmente si la alternativa es una matriz dispersa.

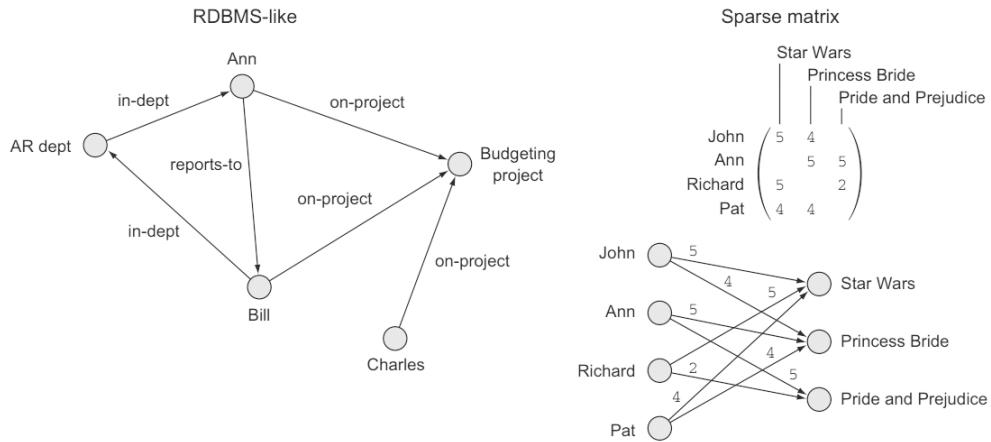


Figura 3.18: RDBMS y matriz dispersa.

3.13.7. Teoría de grafos

La teoría de grafos es un campo de estudio de las matemáticas y las ciencias de la computación, que estudia las propiedades de los grafos. La teoría de grafos es una rama de las matemáticas discretas y de las matemáticas aplicadas, y es un tratado que usa diferentes conceptos de diversas áreas como combinatoria, álgebra, probabilidad, geometría de polígonos, aritmética y topología.

El origen de la teoría de grafos se remonta al siglo XVIII con el problema de los puentes de Königsberg, el cual consistía en encontrar un camino que recorriera los siete puentes del río Pregel en la ciudad de Königsberg, actualmente Kaliningrado, de modo que se recorrieran todos los puentes pasando una sola vez por cada uno de ellos.

La teoría de grafos se usa para la solución de problemas de biología, genética, automatización, entre otros, y ha servido de inspiración para las ciencias sociales, en especial para desarrollar un concepto no metafórico de red social que sustituye los nodos por los actores sociales y verifica la posición, centralidad e importancia de cada actor dentro de la red. Esta medida permite cuantificar y abstraer relaciones complejas, de manera que la estructura social puede representarse gráficamente. Por ejemplo, una red social puede representar la estructura de poder dentro de una sociedad al identificar los vínculos (aristas), su dirección e intensidad y da idea de la manera en que el poder se transmite y a quiénes.

3.13.8. Redes de mundo pequeño (small world networks)

Ocurre en las redes que tienen una conectividad especial que hace que la distancia promedio entre dos actores cualquiera sea muy pequeña en comparación con el tamaño (número de actores) de la red. Un ejemplo de esto es un estudio que realizó Stanley Milgram, un importante psicólogo norteamericano, donde medía la

distancia promedio entre personas, en redes de contacto. Eligió personas de Ohama (Nebraska) y Wichita (Kansas) para que se contactaran con personas de Boston (Massachusetts). La gente de Ohama y Wichita indicaba si conocían, o no, a las personas de Boston. En caso contrario, remitían a un contacto que pudiera conocerlas, con las que se repetía el proceso. Milgram, informado de todo esto, pudo medir cual era el largo de los caminos recorridos. El resultado: 6 personas de distancia en promedio.

3.13.9. Redes libres de escala (scale free networks)

Tienen una distribución de grados que sigue una ley de potencias o similar porque si tomamos un subgrafo de esta red, lo más probable es que los grados se sigan distribuyendo como ley de potencias. Las redes libres de escala son interesantes porque son regidas por leyes de potencia, que se repiten en otros casos como en la distribución del ingreso.

3.13.10. Componente gigante

Una masa o componente conectado es un grupo de nodos y brazos que están todos conectados el uno al otro, ya sea directamente o indirectamente. Si una red tiene un componente gigante conectada, esto significa que la mayoría de los nodos es accesible desde casi todos los demás nodos. Un componente gigante es un grupo de nodos enlazados entre sí, y que agrupan a la mayoría de los nodos de la red.

3.13.11. Distancia geodésica

La distancia geodésica entre un par de nodos en un grafo es la longitud del camino más corto entre los dos nodos, y es la base para definir el diámetro de un grafo. En otras palabras, es la distancia entre dos nodos de un grafo es la longitud del camino más corto.

3.13.12. Medidas de centralidad

La centralidad en un grafo se refiere a una medida posible de un vértice en dicho grafo, que determina su importancia relativa dentro de éste. El concepto tras la posición o localidad de un actor en una red corresponde al acceso que tiene al resto de la red. En principio, sabemos que dos actores ocupan el mismo lugar en la red si comparten los mismo vecinos (equivalencia estructural, una versión local de isomorfismo de vértices). Pero en general deseamos ir más lejos. En esta necesidad definimos las medidas de centralidad, que miden la posición de un actor en una red de acuerdo a ciertos criterios.

3.13.12.1. Centralidad de grado o grado nodal (Degree centrality)

Es el número de actores a los cuales un actor está directamente conectado. Se divide en grado de entrada y salida:

- Grado de salida: es la suma de las relaciones que los actores dicen tener con el resto. Por ejemplo, Pedro dice tener relación con Elohina, Andres y Carlos, por lo cual su grado de salida es 3.
- Grado de entrada: es la suma de las relaciones referidas hacia un actor por otros. Por ejemplo, Israel es mencionado por 4 personas (Rafael, Sebastian, Miguel y Leo), por lo tanto su grado de entrada es de 4.

3.13.12.2. Centralidad de intermediación (Betweenness centrality)

Se interpreta como la posibilidad que tiene un nodo o actor para intermediar las comunicaciones entre pares de nodos. En este análisis se consideran todos los posibles caminos geodésicos entre todos los pares posibles. La medida de intermediación de un nodo se obtiene contando las veces que este aparece en los caminos geodésicos que conectan a todos los pares de nodos de la red, a estos actores se les denomina actores puente.

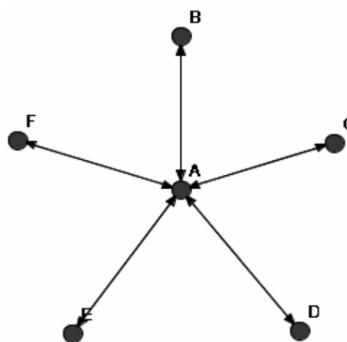


Figura 3.19: Centralidad de intermediación.

En la figura 2.6 podemos visualizar un ejemplo, en donde el nodo A aparece en todos los caminos posibles para que los demás nodos puedan conectarse (F a B, F a C, F a D, F a E, B a C, B a D, B a E, C a D, C a E y D a E), por lo tanto el grado de intermediación de A es 10 y del resto de los actores es 0.

3.13.12.3. Centralidad de vector propio o autovector (Eigenvector centrality)

La centralidad de vector propio mide la influencia de un nodo en una red. Fue propuesta por Phillip Bonacich en 1972, y corresponde al principal vector propio de la matriz de adyacencia del grafo analizado.

Intuitivamente, los nodos que poseen un valor alto de esta medida de centralidad están conectados a muchos nodos que a su vez están bien conectados, también en este sentido; por lo tanto, son buenos candidatos para difundir información, divulgar rumores o enfermedades, entre otros. Los nodos más centrales en este sentido

corresponden a centros de grandes grupos cohesivos. Mientras que en el caso de la centralidad de grado, cada nodo pesa lo mismo dentro de la red, en este caso la conexión de los nodos pesa de forma diferente.

3.13.12.4. Centralidad de Bonacich

Hay gente cuya favorable posición en una red social les permite iniciar procesos influenciales como la transmisión de creencias, chismes (gossip), publicidad viral, etc. En estos casos, el proceso empieza en un actor y se distribuye a su vecinos, los cuales redistribuyen a sus propios vecinos, sucesivamente. Entonces, podemos proponer una medida de centralidad para tales situaciones, que consista en contar los caminos. Sin embargo, las redes con ciclos nos dan problemas pues tienen infinitos caminos. Por eso, no podemos contar los caminos así nada más. La solución práctica a este dilema es atenuar los caminos usando una tasa de descuento, tal como se usa en la evaluación económica, las series de potencias, etc.

Así, los caminos más largos se suman como números más pequeños, y los infinitos son cero. Usar una tasa de descuento que hace más pequeños los caminos más grandes tiene ventajas conceptuales. Los procesos influenciales como los chismes, las creencias, etc. pueden ser muy efectivos en distancias cortas, pero su difusión se hace menos efectiva (o lenta) a grandes distancias. Ajustando la tasa de descuento, se puede simular cuán rápido se atenúa un proceso de difusión.

3.13.13. Detección de comunidades

La detección de comunidades, grupos, cliques (grupos exclusivos), entre otros, es tema de alto interés en las redes. El asunto es complicado pues no es fácil definir un grupo. La definición es fácil cuando hablamos de una estructura formal, cuando existe un grupo definido y un grupo de adherentes que dice ser parte del grupo. Por ejemplo, Chile y los chilenos. Pero todo se vuelve complicado, oscuro, hasta esotérico cuando hablamos de la estructura informal.

Existen muchas técnicas y algoritmos para detectar comunidades, con muchas velocidades diferentes, que obedecen a diferentes ideas de cómo se detecta un grupo, situación muy opuesta a la de las medidas de centralidad. Una manera tradicional consiste en reducir la detección de grupos a una clasificación o clustering. Dentro de estas técnicas están el tradicional k-Means, los algoritmos genéticos, el análisis de modularidad (el número de vínculos entre grupos pequeño, dentro de grupos es alto), etc. Adicionalmente, estas técnicas son parametrizables (i.e. número de clases en k-Means, modularidad mínima, etc.), lo que permite analizar la calidad de la clasificación. Aquí se hace posible usar árboles jerárquicos para decidir cuándo la clasificación es buena. Algunos métodos y algoritmos para la detección de comunidades en los grafos, son las siguientes:

- Métodos jerárquicos:

- Aglomerativos.
- Newman-Girvan.
- Radicchi.
- Métodos modulares:
 - Modularidad Q.
 - Algoritmo greedy.
 - Algoritmo Fast Greedy.
 - Algoritmo MultiStep Greedy.
- Métodos particionales:
 - Cliques y k-cores.
 - N-cliques.
 - K-Means.
 - Kernighan-Lin.
- Métodos espectrales:
 - Conductancia.
 - Bisección espectral (EIG1).
 - UKMeans.
 - Algoritmo NJW.

Otra manera tradicional consiste en ver el problema como uno de teoría de grafos. Por ejemplo, la coloración es una forma tradicional de hacer clasificación en grafos. En este caso, también es posible ver el problema como uno de equivalencia estructural transformado a uno de equivalencia regular: "en un grupo de amigos, los amigos compartimos los mismos amigos" (esto define un algoritmo iterativo). Adicionalmente, se pueden buscar cliques y k-Cliques para encontrar los grupos.

3.13.14. Principales V's de Big Data para grafos

- Volumen: es cuando el crecimiento o el tamaño del grafo es más grande que la cantidad de memoria disponible.
- Velocidad: se refiere a las actualizaciones en tiempo real del grafo tomando en cuenta la velocidad en la que puede crecer.
- Variedad: se debe tomar en cuenta la variedad, que se puede clasificar en 2 tipos, la fuente de datos, como las base de datos relacionales, XML, JSON, documentos y otros, y por el significado o la naturaleza del grafo, por ejemplo las redes sociales, biológicas, de información, entre otros. Este ultimo es importante, ya que hay casos en el que se pueden hacer inferencias, "mi mascota es

un perro, mi perro es un mamífero, entonces mi mascota es un mamífero", esta inferencia tiene sentido, pero existen otros casos en los que no, por ejemplo con las proteínas y genes.

- Valencia: se refiere a la conectividad entre los nodos, si aumentamos la valencia de un grafo, se aumenta la conectividad de este.

3.14. Bulk Synchronous Parallel (BSP)

Bulk Synchronous Parallel es un modelo o abstracción que permite diseñar algoritmos para que se ejecuten en paralelo. Fue desarrollado por Leslie Valiant de la Universidad de Harvard en 1980. El artículo definitivo fue publicado en 1990. BSP consiste en:

- Tener componentes capaces de procesar transacciones en memoria local.
- Tener una red que enrute los mensajes entre pares de dichos componentes.
- Tener un hardware que permita la sincronización de todos o de un subconjunto de los componentes.

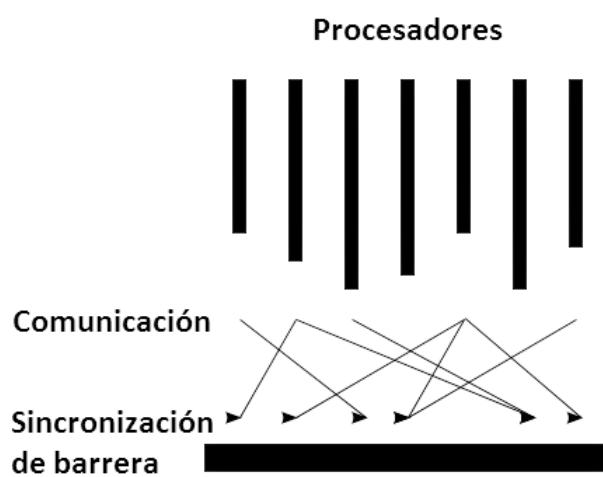


Figura 3.20: Bulk Synchronous Parallel.

Esto se interpreta generalmente como un conjunto de procesadores que pueden seguir diferentes hilos (threads) de computación, con cada procesador equipado con memoria local rápida e interconectados por una red de comunicación. Un algoritmo BSP se basa en gran medida en la tercera característica; un cálculo procede de una serie de superetapas globales, que consta de tres componentes:

- Computación concurrente: cada procesador participante puede realizar cálculos locales, es decir, cada proceso sólo puede hacer uso de los valores almacenados en la memoria rápida local del procesador. Los cálculos se producen de forma asíncrona sobre todos los demás, pero pueden superponerse con la comunicación.
- Comunicación: Los procesos realizan cambios de datos entre sí para facilitar la capacidad de almacenamiento de datos a distancia.
- Sincronización de barrera (Barrier synchronisation): Cuando un proceso llega a este punto (la barrera), se espera hasta que todos los demás procesos han llegado a la misma barrera.

3.14.1. Pregel

Pregel es un modelo de programación inspirada en BSP, orientada específicamente a los problemas de grafos de gran escala. Es útil por dos razones:

- Facilidad de programación. El modelo de programación es natural cuando se trabaja con grafos, ya que hace vértices y aristas sean clases, lo que anima a los programadores a pensar en esos términos, en lugar de en términos de flujos de datos y operadores de transformación en partes de los grafos.
- Eficiente en los problemas de los grafos. Está diseñado para apoyar los cálculos iterativos más eficientemente que MapReduce, ya que mantiene el conjunto de datos en la memoria en lugar de escribir en el disco después de cada iteración. Esto es útil ya que muchos algoritmos de grafos son iterativos.

Pregel es un modelo de programación que fundamentalmente utiliza el paso de mensajes entre los vértices de un grafo, organizados en una secuencia de iteraciones llamado superpasos (supersteps). Al comienzo de cada superetapa, se ejecuta una función de cálculo especificado por el usuario en cada vértice, pasándole todos los mensajes enviados a ese vértice durante la última superetapa.

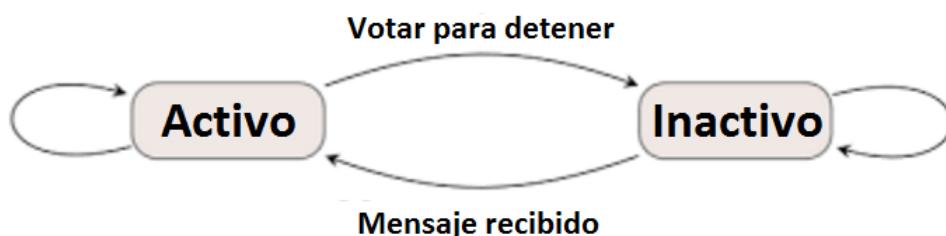


Figura 3.21: Pregel.

Divide un grafo en particiones, cada uno compuesto de un conjunto de vértices y todos bordes salientes esos vértices. La función de cómputo tiene la oportunidad de

procesar estos mensajes y enviar mensajes a otros vértices, que serán recibidos en la siguiente superetapa. También puede "votar para detener", desactivando el vértice que se está ejecutando hasta el vértice reciba un mensaje. Una vez que todos los vértices han votado para poner fin, termina el trabajo.

La ventaja de este modelo es que una vez que se comience a "pensar como un vértice." enviar mensajes a otros vértices, los problemas de grafos de gran escala se hacen mucho más fácil. Por ejemplo, una implementación de ejemplo de PageRank en Pregel está a sólo 15 líneas de código C++.

3.15. Sistemas para grafos (Graph systems)

Los sistemas de grafos pueden ser divididos en dos categorías, sistemas de procesamiento y de almacenamiento. Algunos están basados en memoria y soportan el manejo de clusters.

3.15.1. Sistemas de almacenamiento

Las bases de datos orientadas a grafos gracias a los lenguajes de consultas son convenientes para conseguir información sobre algún vértice o conjunto de aristas en particular, los sistemas basados en Pregel para el procesamiento de grafos, en contraste, no son buenos realizando consultas pero son buenos ejecutando masivamente algoritmos en paralelo, por ejemplo PageRank. Las bases de datos orientada a grafos poseen la ventaja de que proveen una base de datos de transacciones, lenguajes de consultas y un fácil manejo para las actualizaciones y persistencia, sin embargo si están basadas en disco, no tienen un buen desempeño para procesar grafos. Los grafos pueden ser almacenados en sistemas de almacenamiento de archivos distribuidos y/o en base de datos orientadas a grafos.

3.15.1.1. Neo4j

Neo4j es una base de datos orientada a grafos escrita en Java, es decir la información se almacena de forma relacionada formando un grafo dirigido entre los nodos y las relaciones entre ellos. Neo4j fue desarrollado por Neo Technology, una startup sueca con base en Malmö y San Francisco Bay Area en Estados Unidos. La versión 1.0 de Neo4j fue lanzada en febrero de 2010. [47]

Se integra perfectamente con múltiples lenguajes como Java, PHP, Ruby, .Net, Python, Node, Scala, etc. La base de datos está embebida en un servidor Jetty. Está especialmente indicada para modelar redes sociales y sistemas de recomendación. Se distribuye en dos versiones: la community edition (open source) y la enterprise edition. Debido a que GraphX posee un veloz procesamiento, puede asociarse con base de datos orientada a grafos como Neo4j, y utilizar lo mejor de los 2 mundos: base de datos de transacciones y la velocidad de procesamiento.

3.15.1.2. Titan

Titan es una base de datos orientada a grafos, bajo licencia Apache 2 escalable y optimizada para almacenar y consultar grafos muy grandes con billones de vértices. Sus principales características son:

- Escalabilidad lineal.
- Soporte ACID.
- Distribución de datos y replicación enfocada al rendimiento y tolerancia a fallos.
- Soporte para almacenamiento en diferentes backends como Apache Cassandra, Apache HBase, Oracle BerkeleyDB, entre otros.

3.15.1.3. HDFS

HDFS es un sistema de ficheros distribuido, escalable y portátil escrito en Java y creado especialmente para trabajar con ficheros de gran tamaño. (Ver más en el capítulo 3.12.2).

3.15.1.4. DEX/Sparksee

Sparksee (anteriormente DEX) es una base de datos orientada a grafos escrita en C++ que permite analizar grandes volúmenes de datos. DEX está basado en el modelo2 de base de datos en grafo, que está caracterizado por cumplir 3 propiedades: las estructuras de los datos son grafos o estructuras similares a un grafo, la manipulación de los datos y las consultas se realizan con operaciones orientadas a grafo y existen restricciones para garantizar la integridad de los datos y de sus relaciones. [48]

Su desarrollo empezó en el 2006 como un producto originado de la investigación de DAMA-UPC (grupo Data Management de la Universidad Politécnica de Catalunya (UPC), Barcelona). La primera versión estaba disponible en el tercer cuatrimestre del 2008. En marzo de 2010 nació la empresa Sparsity Technologies creada desde la UPC para comercializar y dar servicios a las tecnologías desarrolladas en DAMA-UPC.

En febrero de 2014 para la quinta versión de la base de datos, DEX cambia su nombre a Sparksee. Hay diversas versiones gratuitas de licencias para uso personal, de investigación o desarrollo. La principal característica de Sparksee es su capacidad de almacenamiento de datos y rendimiento, con órdenes de magnitud de miles de millones de nodos, aristas y atributos, gracias a una implementación con estructuras ligeras especializadas.

Un grafo de Sparksee es un multigrafo dirigido etiquetado y con atributos. Está etiquetado porque tanto nodos como aristas pertenecen a tipos. El grafo es dirigido

porque permite que existan tanto aristas dirigidas como no dirigidas. Nodos y aristas pueden tener tantos atributos como se deseé. Finalmente decimos que es un multigrafo porque permite que existan múltiples aristas entre dos nodos aunque éstas sean del mismo tipo.

3.15.1.5. Oracle Spatial

Oracle Spatial es un componente opcional de las base de datos de Oracle. Es compatible con una amplia gama de datos geoespaciales y analíticas para la gestión de los servicios de localización móviles, gestión de territorio de ventas, transporte, entre otros. Las funciones de los grafos incluyen grafos RDF para aplicaciones que van desde la integración semántica de datos, análisis de redes sociales, grafos de red utilizados en el transporte, servicios públicos, energía, empresas de telecomunicaciones y análisis en tiempo real para las aplicaciones comerciales y de marketing. [49]

3.15.2. Sistemas de procesamiento

Existen múltiples sistemas para el procesamiento de grafos de gran escala, donde cada uno tiene distintos métodos de almacenamiento, particionamiento y procesamiento.

3.15.2.1. Ringo

Ringo es un sistema para el análisis de datos interactivo para flujos de trabajo que implican representaciones de datos como grafos. Ringo proporciona un entorno de alta productividad para la construcción, análisis y manipulación de grafos en una sola máquina multi-núcleo de gran memoria, que combina el análisis de alta productividad con tiempos de ejecución rápidas y escalables. [50]

Ringo ofrece las siguientes características:

- Una interfaz Python interactiva y fácil de usar.
- Un amplio conjunto de más de 200 operaciones avanzadas y algoritmos (basado en la librería de grafos SNAP) para grafos.
- Integración y procesamiento de grafos, apoyo para la construcción de grafos de forma eficiente y transformaciones entre tablas y grafos.
- Un amplio conjunto de más de 200 operaciones con gráficos avanzados y algoritmos (basado en la librería gráfica SNAP).

La orientación de las máquinas individuales de gran memoria ofrecen muchos beneficios para el análisis interactivo, en comparación con los entornos distribuidos, tanto en términos de rendimiento y facilidad de programación. En Ringo las operaciones por tabla, transformaciones entre tablas y grafos, y varios algoritmos de

grafos son totalmente paralelizados para aprovechar al máximo el ambiente multi-núcleo, y el conjunto de algoritmos de grafos están disponibles para la ejecución en paralelo bajo constante expansión.

3.15.2.2. Pegasus

Pegasus es un sistema de minería de grafos basado en Peta-escala, escrito en Java. Se ejecuta en forma paralela y distribuida sobre Hadoop. [51] Provee algoritmos para grafos de gran escala que son importantes en minería de grafos, por ejemplo:

- Degree.
- PageRank.
- Random Walk with Restart (RWR).
- Radius.
- Connected Components.

Pegasus rompe los límites de los algoritmos para grafos que tienen una escalabilidad limitada, permitiendo el uso de grafos de gran escala gracias a Hadoop, representándolos como matrices y procesándolos con MapReduce.

3.15.2.3. Neo4j Mazerunner

Mazerunner es una extensión de Neo4j y una plataforma de procesamiento de grafos distribuido donde Neo4j puede realizar grandes trabajos de procesamiento de grafos utilizando Apache Spark, donde los resultados son traídos de vuelta a Neo4j. [52] Mazerunner utiliza un intermediario de mensajes para distribuir trabajos de procesamiento de grafos, GraphX que es un módulo de Apache Spark. Cuando se envía un trabajo, un subgrafo se exporta desde Neo4j y se escribe en Apache Hadoop HDFS.

Luego un servicio independiente de Mazerunner notifica a Spark que puede empezar a procesar esos datos. El servicio Mazerunner a continuación, notifica la iniciación de un algoritmo de procesamiento de grafos distribuido utilizando el módulo GraphX, junto con Scala y Spark. El algoritmo GraphX es serializado y enviado a Apache Spark para su procesamiento.

Una vez que el trabajo de Apache Spark este completa, los resultados se vuelven a escribir en HDFS como una lista de clave-valor junto con una de cambios o actualizaciones que debe aplicarse en Neo4j. Neo4j notifica entonces que una lista de actualización de propiedades está disponible. Neo4j importa los resultados y aplica las actualizaciones de nuevo al grafo original.

3.15.2.4. Graphlab/PowerGraph

GraphLab ahora llamado PowerGraph es un framework de procesamiento de grafos distribuidos de alto rendimiento escrito en C++. [53] El proyecto se inició por el Prof. Carlos Guestrin de la Universidad Carnegie Mellon en 2009. Se trata de un proyecto de código abierto utilizando una licencia Apache inspirado en las ideas del artículo de Google Pregel. Mientras GraphLab fue desarrollado originalmente para tareas de aprendizaje automático, se ha encontrado un gran éxito en una amplia gama de otras tareas de minería de datos. Las principales consideraciones de diseño detrás de GraphLab son:

- Datos dispersos mediante dependencias locales.
- Algoritmos iterativos.
- Basado en ejecución asíncrona.

Algunas de sus características son:

- Integración con HDFS.
- Kits de herramientas de aprendizaje automático de gran alcance.
- Escalable: GraphLab inteligentemente almacena los datos mediante algoritmos sofisticados.
- Posee un multi-núcleo unificado y un API distribuido.

3.15.2.5. Apache Giraph

Apache Giraph es un sistema de procesamiento de grafos interactivo construido para una alta escalabilidad. Se originó como la contraparte de Pregel, la arquitectura de procesamiento de grafos desarrollado en Google y descrito en un documento de 2010, ambos sistemas inspirados en el modelo distribuido Bulk Synchronous Parallel.[54]

Facebook utiliza Giraph con algunas mejoras de rendimiento para analizar un billón de aristas utilizando 200 máquinas en 4 minutos. Giraph es limitado debido a lo lento que es usar Hadoop MapReduce.

3.15.2.6. GraphX

GraphX es un API de Apache Spark para grafos y su procesamiento paralelo, cuyas implementaciones están basadas en el artículo de Google Pregel. GraphX fue incorporado en Spark 1.0 el 30 de Mayo del 2014, sin embargo el 9 de Febrero del 2015 (Spark 1.2.1) se añadieron cambios importantes como la funcionalidad aggregateMessage y el tipo de dato EdgeRDD[ED]. En el capítulo 4 se desarrolló más a fondo distintos aspectos de este API.

3.16. Herramientas para la visualización de grafos

La visualización de las redes también sirve como método para descubrir propiedades de ésta, aunque tiene menos peso teórico en el análisis, tiene la ventaja de alimentar rápidamente la intuición del investigador. Visualizar redes complejas es un gran desafío; por lo general, se busca presentar gran cantidad de información de forma estética. Se busca la claridad y la simpleza, pese a la gran complejidad de los datos.

Hay que considerar que hay muchas potenciales vistas de los datos, que pueden ilustrar propiedades diferentes: centralidad, comunidades, jugadores clave (que, si desaparecen, desconectan la red), entre otros. Existe una gran variedad de algoritmos para visualizar redes, cada uno obedece a una idea u objetivo diferente, aunque muchas veces se busca la presentación instantánea. Existen múltiples herramientas que nos permiten visualizar redes o grafos de todo tipo, algunas de estas son las siguientes.

3.16.1. Gephi

Gephi es una herramienta de licencia libre desarrollada en Java para visualizar y analizar grandes grafos. Usa un motor de renderizado 3D para mostrar gráficos en tiempo real y permite explorar, analizar, filtrar, clusterizar, manipular y exportar diversos tipos de gráficos. Tiene la particularidad que permite manejar grafos grandes (de miles de nodos) con un muy buen desempeño, cosa que no es muy común en este tipo de herramientas. [55]

Permite agrupar nodos del grafo, pintarlos de diferentes colores, modificar tamaños, hacer las aristas entre nodos más gruesas dependiendo de diversos factores, entre otros.

3.16.2. Graphviz

Graphviz es una aplicación de visualización de grafos de código abierto que incluye un gran número de programas de trazado de grafos; además cuenta con interfaces interactivas y vía web, así como herramientas auxiliares y bibliotecas de funciones, existiendo versiones tanto para Windows como para Linux. Los programas de diseño de Graphviz parten de descripciones de grafos en texto plano, lo que les permite ser editados por los usuarios y no necesitar un programa adicional para ello. Los diagramas son realizados en varios formatos: imágenes (jpg o png), SVG (Scalable Vector Graphics, gráficos vectoriales en dos dimensiones) para páginas web, Postscript para ser incluido en PDFs u otros documentos, o también pueden ser representados en un navegador interactivo, donde el usuario pueda editarlos (Graphviz también soporta GXL, Graph eXchange Language). Graphviz cuenta con muchas características para personalizar los diagramas tales como opciones para etiquetas, colores, fuentes,

diseños en forma de tabla, estilos de línea, enlaces y formas. [56]

3.16.3. D3.js

D3.js (o simplemente D3 por las siglas de Data-Driven Documents) es una librería de JavaScript distribuida en Agosto de 2011 [57], para producir, a partir de datos, infogramas dinámicos e interactivos en navegadores web. Básicamente, la librería permite manipular documentos basados en datos usando estándares abiertos de la web y los navegadores pueden crear visualizaciones complejas sin depender de un software propietario. Hace uso de tecnologías bien sustentadas como SVG, HTML5, y CSS. Esta librería es sucesora de la librería Protovis. En contraste con muchas otras librerías, D3.js permite tener control completo sobre el resultado visual final.

3.16.4. GraphStream

GraphStream es una librería de Java para el modelado y análisis de grafos de forma dinámica. Su finalidad se centra en el modelado de las redes de interacción dinámica de varios tamaños, proporcionando una forma de representar grafos. GraphStream proporciona una manera de manejar la evolución de los grafos en el tiempo. Esto significa que manipula la forma en que los nodos y aristas se añaden y eliminan, y como los atributos de los datos aparecen, desaparecen y evolucionan. [58]

CAPÍTULO 4

GraphX

GraphX es el nuevo API de Spark de computación paralela para grafos, en otras palabras es Spark aplicado a los grafos. Anteriormente para el manejo de grafos de gran escala las personas utilizaban el módulo de Spark, Bagel, pero actualmente con GraphX existe una forma estandarizada para hacerlo, y también proporciona una biblioteca de algoritmos útiles. Al representar datos que poseen conexiones como un grafo, se puede utilizar una serie de herramientas y técnicas para obtener información de esto. Por ejemplo PageRank y coeficiente de agrupamiento.

Estas son algunas razones por las que utilizar GraphX Spark es ideal:

- La organización ya posee Spark y desea incorporar procesamiento de grafos.
- El grafo es demasiado grande para ser almacenado y procesado en una sola máquina.
- La organización le gusta ser concisa utilizando el lenguaje Scala.
- Desea utilizar una base de datos de grafos como Neo4j y Titan en conjunción con GraphX.

GraphX no es una base de datos, por lo tanto no se ocupa de las actualizaciones y eliminaciones como Neo4j y Titan, es un sistema de procesamiento de grafos, que es útil para diversas aplicaciones que pueden ser complicados ejecutar en una sola máquina, por ejemplo extraer información de conjuntos de datos conectados puede aportar numerosas soluciones, como la detección de fraude y la clasificación de las páginas webs. Las implementaciones de GraphX, están basadas en las propiedades de los grafos, donde representa a los vértices y aristas con 2 RDDs, y gracias a esto, pueden particionarse grafos de gran escala.

Este API es todavía joven, y posee algunas limitaciones, la mayoría de estas son las mismas que posee Spark. Por ejemplo, los datasets de GraphX, como los de Spark, normalmente no pueden ser compartidos, es limitado por la inmutabilidad de Spark RDD, lo que es un problema para grafos de gran escala y en muchos casos es más lento que otros sistemas, como GraphLab/PowerGraph, debido a su dependencia con JVM.

4.1. Particionamiento de grafos de gran escala

Si se tiene un grafo muy grande para ser almacenado en la memoria de un solo ordenador, Spark permite dividirlo entre varios ordenadores en un grupo de clusters, pero ¿cuál es la mejor manera de dividir un grafo?.

La forma ingenua, y la forma en que se ha hecho durante muchos años, fue asignar diferentes vértices a los diferentes clusters, pero esto condujo a los cuellos de botella en los cálculos, porque los grafos del mundo real poseen una distribución similar a una ley de potencia y suelen tener vértices con altos grados.

La partición de grafos por vértices se llama edge-cut, porque en realidad se cortan son las aristas, sin embargo, la mayoría de los sistemas de procesamiento de grafos emplean el método vertex-cut, que distribuye uniformemente las aristas entre las máquinas nodos, de manera más balanceada en los clusters. Vertex-cut fue popularizada por GraphLab (ahora llamada PowerGraph), y fue adoptada por GraphX como el esquema de partición por defecto.

4.2. Grafos o datos en paralelo. (Graph parallel or data parallel)

GraphX deja escoger entre la utilización de grafos en paralelo o datos en paralelo, facilita el acceso a los datos y a diversas operaciones. Almacena las aristas en un tabla y los vértices en otra, esto permite que distintos algoritmos recorran de forma eficiente los grafos mediante las aristas de un vértice a otro.

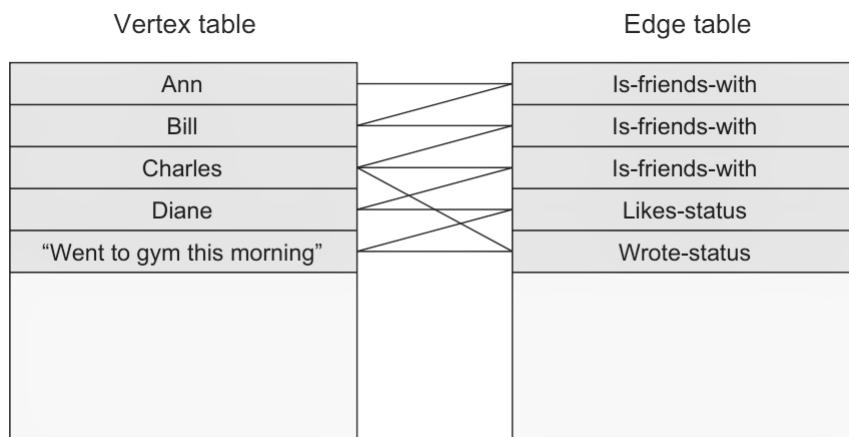


Figura 4.1: Grafos o datos en paralelo.

Aunque GraphX almacena las aristas y los vértices en tablas separadas, puede utilizar un esquema RDBMS, e internamente existen índices especiales para recorrer

rápidamente el grafo, y proporciona un API que facilita las consultas sobre grafos similar a SQL.

4.3. Tipos de flujos de datos

GraphX es inherentemente un sistema de procesamiento por lotes, que no se integra con Spark Streaming, por ejemplo (al menos no de una manera sencilla). No hay una sola manera de utilizar GraphX, existen muchos tipos de datos de procesamiento por lotes.

Debido a las capacidades de GraphX para la lectura de archivos de grafos limitados, los archivos de datos por lo general tienen que ser procesados mediante Spark Core API en el formato de grafos que utiliza GraphX. Las salida de los algoritmos pueden ser otro grafo, un número, subgrafos, o un modelo de aprendizaje automático.

4.4. Almacenamiento de grafos

Debido a que GraphX es estrictamente un sistema de procesamiento en memoria, es necesario tener un lugar para almacenar los grafos. Spark suele usar sistemas de almacenamiento distribuidos como HDFS o S3, sin embargo GraphX se puede conectar con base de datos para grafos.

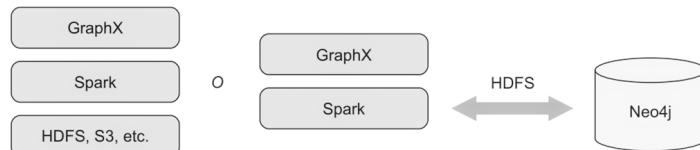


Figura 4.2: Almacenamiento de grafos.

La forma convencional y más común es almacenar los datos son los sistemas de almacenamiento distribuido, principalmente HDFS. Algunas organizaciones utilizan una base de datos de grafos, y utilizan las ventajas de ambos, las transacciones en una base de datos grafos y el procesamiento rápido de GraphX.

4.5. Algoritmos para grafos

4.5.1. PageRank

El algoritmo de PageRank, fue inventado originalmente para establecer posiciones o rangos a las páginas de los buscadores, pero puede ser usado para medir influencia en las redes sociales, en las citas bibliográficas, entre otros. En otras palabras, puede ser usado para encontrar los nodos importantes de cualquier grafo.

PageRank fue patentado por la Universidad de Stanford y posee una marca registrada de Google.

Aunque la aplicación original de PageRank fue asignar un número de influencia para cada página web en un motor de búsqueda, puede ser utilizado en cualquier grafo dirigido para establecer la influencia de cada nodo en el grafo. Este algoritmo se puede describir en los siguientes pasos:

1. Inicializar los vértices en $1/N$, donde N es el número de vértices del grafo.
2. Loop:
 - a) Para cada vértice, transmitir un PageRank de $1/M$ a cada arista de salida, donde M es el grado de salida del vértice.
 - b) En cada vértice que recibe un PageRank de sus vértices adyacentes, sumar los valores y hacer que este sea el nuevo PageRank del vértice.
 - c) Si el PageRank no ha cambiado significativamente a través del grafo desde la iteración anterior, entonces el algoritmo finaliza.

4.5.2. Contador de triángulos (Triangle Count)

Mide la conectividad de un grafo o un subgrafo. Por ejemplo, en una red social, si todo el mundo influye en los demás, si todo el mundo está conectado con el resto, habrá una gran cantidad de triángulos. Un triángulo es cuando tres vértices están conectados por aristas entre sí. Mientras más triángulos tenga un grafo, más conectado se encuentra. [59]

Al momento de contar triángulos GraphX ignora las aristas que poseen dirección, es decir que las maneja como si no tuvieran y de esta forma se eliminan los bucles, y las aristas bidireccionales se tratan como una sola. La propiedad mientras más triángulos posea un grafo, es usada en múltiples casos, por ejemplo en la detección de tipos de clicks para identificar spammers y realizar recomendaciones. Triangle Count sirve como un factor en otras dos métricas conocidas como el coeficiente de agrupamiento y las relaciones de transitividad, sin embargo a partir de la versión 1.6, GraphX no posee estos 2 algoritmos, debido a que es complicado procesar la cantidad de triángulos que generan.

4.5.3. Caminos cortos (Shortest Paths)

La implementación de este algoritmo en GraphX cuenta el número de saltos, es decir que devuelve la distancia en términos de número de saltos.

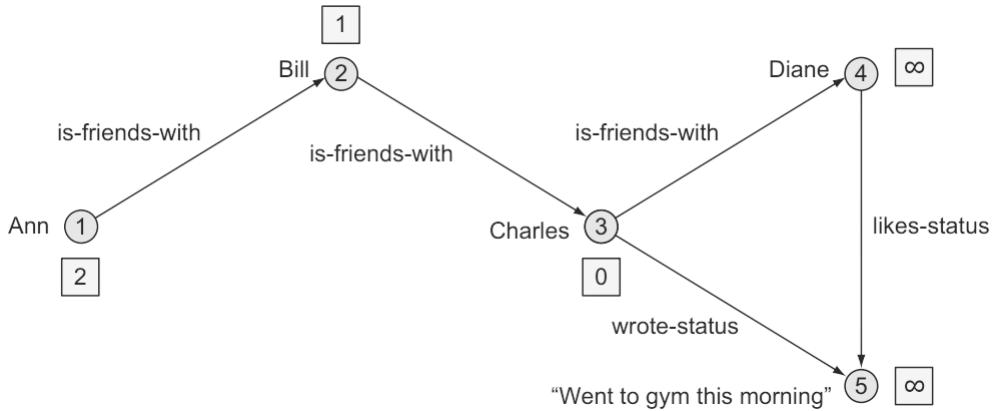


Figura 4.3: Shortest Paths.

Este algoritmo encuentra el número de saltos de cada vértice a un vértice en particular. No toma en cuenta algún tipo de peso que representen las aristas, y sólo devuelve el número de saltos. En la figura 4.3 podemos ver un ejemplo, donde los cuadrados representan el número de saltos, y el vértice particular es el 3 (Charles).

4.5.4. Componentes conectados (Connected Components)

Un componente conectado es un conjunto de vértices en un grafo que están vinculados entre sí por caminos. Pueden ser útiles para hallar clicks y particionar un centro de datos. Este es relevante para grafos dirigidos y no dirigidos.

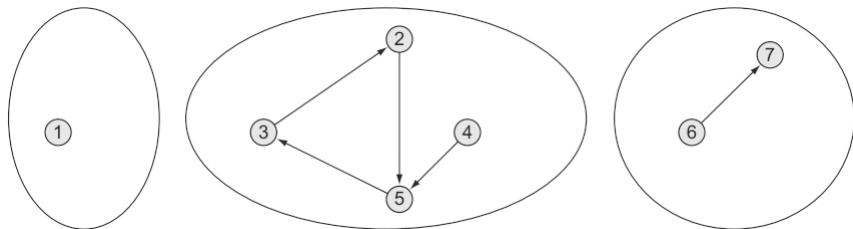


Figura 4.4: Connected Components.

GraphX cuando ejecuta este algoritmo recibe un grafo y retorna otro con la misma estructura que el anterior. En la figura 4.4 podemos observar que hay 3 componentes.

4.5.5. Componentes fuertemente conectados (Strongly Connected Components)

Para los grafos dirigidos, a veces se requiere eliminar los callejones sin salida de los componentes. En las redes sociales, los componentes fuertemente conectados pueden servir de base para un motor de recomendación si se añaden otros aspectos al motor. En este algoritmo cada vértice es accesible desde otro que se encuentre

en un componente. Dentro de un componente fuertemente conectado, ningún vértice puede actuar como un callejón sin salida (dead end).

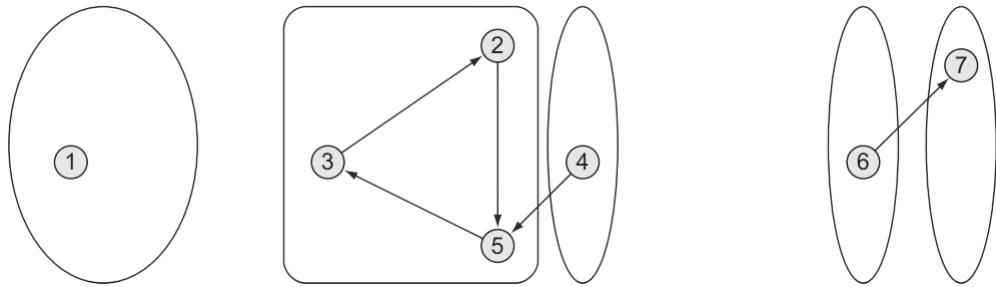


Figura 4.5: Strongly Connected Components.

Otra aplicación es garantizar que en una máquina de estados no hay callejones sin salida, donde la máquina de estado puede llegar a atascarse. También son útiles en la construcción de compiladores de optimización para cuando se hacen análisis de flujos de datos para identificar expresiones que no se acostumbran a usar y que de otro modo sean un desperdicio computacionalmente.

4.5.6. Propagación por etiqueta (Label Propagation)

Para identificar comunidades en un grafo, GraphX provee el algoritmo de Label Propagation descrito por Raghavan en el 2007, en su artículo “Near linear time algorithm to detect community structures in large-scale networks”.

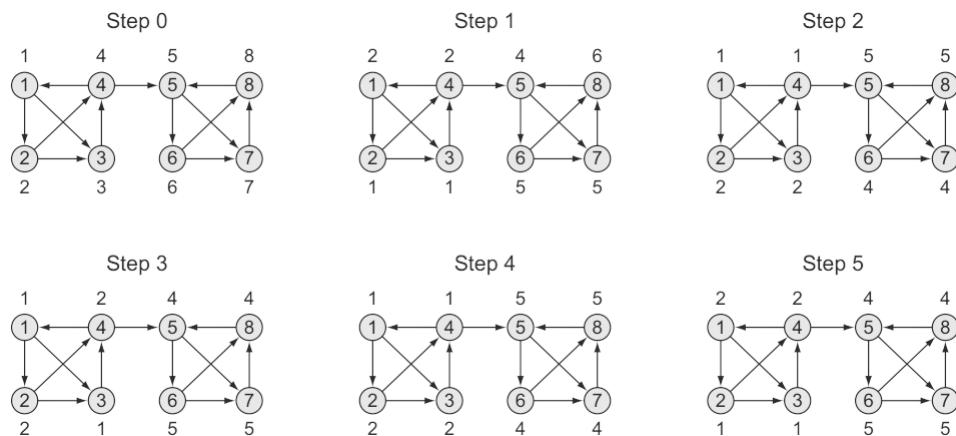


Figura 4.6: Label Propagation.

La idea es tener grupos grandes de vértices conectados que formen conjuntos bajo una misma y única etiqueta que define la comunidad. GraphX sólo proporciona una versión estática de este algoritmo, que se ejecuta durante un número de iteraciones que se tiene que especificar. El problema de este algoritmo es que es poco útil ya que rara vez converge. La propagación por etiqueta es un algoritmo para

encontrar comunidades. En comparación con otros algoritmos tiene como ventaja un bajo tiempo de ejecución. Posee como desventaja que no produce una solución única, sino un agregado de muchas soluciones.

Los nodos tienen una etiqueta que indica la comunidad a la que pertenecen. La pertenencia a una comunidad cambia, a partir de las etiquetas que los nodos vecinos poseen. Este cambio está sujeto a la cantidad máxima de las etiquetas dentro de un grado de los nodos. Cada nodo se inicializa con una etiqueta única, luego las etiquetas se difunden a través de la red. En consecuencia, los grupos densamente conectados alcanzan una etiqueta común rápidamente. Este algoritmo es considerado semi-supervisado y es similar a K-vecinos más cercanos (KNN).

4.5.7. Dijkstra

Este algoritmo busca el camino más corto desde un vértice a otro utilizando aristas que poseen peso o ponderación.

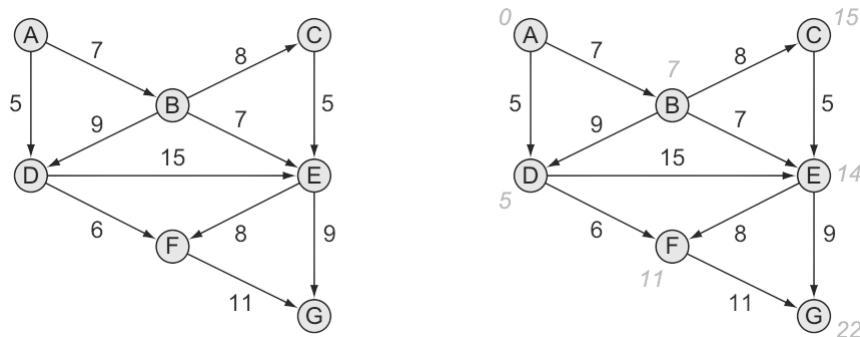


Figura 4.7: Dijkstra.

La idea subyacente en este algoritmo consiste en ir explorando todos los caminos más cortos que parten del vértice origen y que llevan a todos los demás vértices; cuando se obtiene el camino más corto desde el vértice origen, al resto de vértices que componen el grafo, el algoritmo se detiene. El algoritmo es una especialización de la búsqueda de costo uniforme, y como tal, no funciona en grafos con aristas de coste negativo (al elegir siempre el nodo con distancia menor, pueden quedar excluidos de la búsqueda nodos que en próximas iteraciones bajarían el costo general del camino al pasar por una arista con costo negativo).

4.5.8. Problema del agente viajero (Travelling Salesman)

Este algoritmo trata de encontrar el camino más corto a través de un grafo no dirigido. Por ejemplo, si un vendedor tiene que visitar cada ciudad en una región, le gustaría minimizar la distancia total recorrida. Una implementación es un algoritmo voraz (greedy algorithm), que es el algoritmo más simple pero también da respuestas que pueden ser poco óptimo.

CAPÍTULO 5

Marco metodológico

Para implementar la solución de un problema es importante tener un orden y establecer distintas prioridades, es decir una metodología. Realizar análisis sobre grafos de gran escala puede ser complejo, por lo tanto realizar un conjunto de actividades estructuradas y metódicas es de gran ayuda.

Específicamente para este trabajo de investigación se decidió utilizar la Metodología Fundamental para la Ciencia de Datos propuesta por la empresa IBM. Consiste en los siguientes pasos:

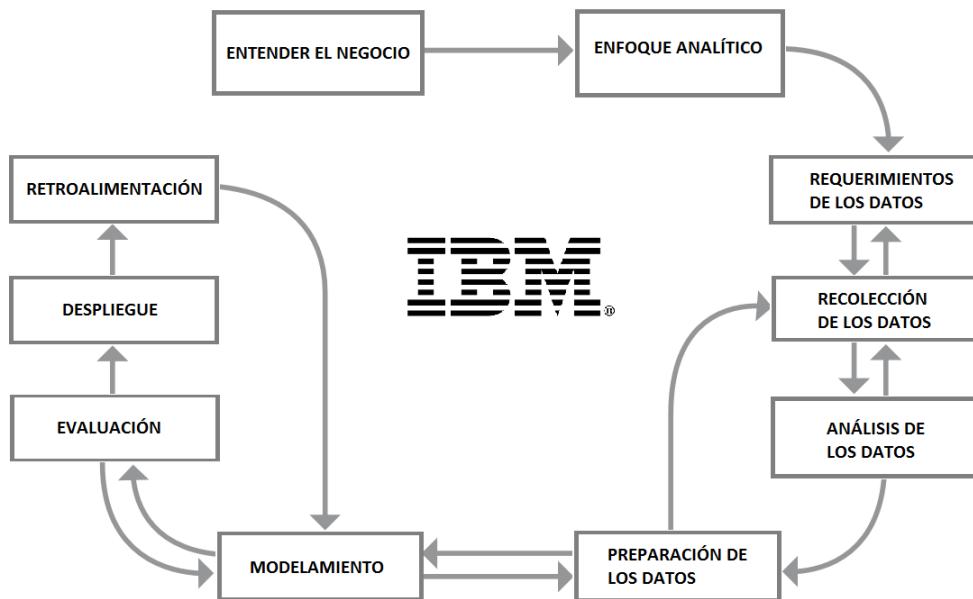


Figura 5.1: Metodología para Ciencia de los Datos.

- Entender el negocio: Todo proyecto, independientemente de su tamaño, comienza con la comprensión del negocio y establecer las bases para una solución exitosa del problema. Los usuarios finales deben plantear los objetivos del proyecto y los requisitos de la solución, por lo tanto es de suma importancia una fase de análisis.

- Enfoque analítico: Una vez que el problema de la empresa se ha establecido claramente, el científico de datos puede definir el enfoque analítico para resolver el problema. Es decir, expresar el problema en un contexto basado en técnicas estadísticas, algoritmos e infraestructura para que el científico de datos puede identificar las técnicas adecuadas para lograr el resultado deseado.
- Requerimientos de los datos: El enfoque analítico elegido determina los requisitos de los datos . En concreto, los métodos analíticos que se utilizan requieren un conjunto de datos, formatos y representaciones, guiados por el conocimiento basados en el dominio.
- Recolección de los datos: En la etapa inicial de la recopilación de datos, los científicos de datos identifican y reúnen los recursos o datos estructurados, no estructurados y semi-estructurados que sean relevantes para el problema. Por lo general, tienen que elegir si desea realizar inversiones adicionales para obtener datos menos accesibles o con características particulares. Si faltan datos en el proceso de recopilación, el científico de datos puede tener que revisar los requerimientos de estos en consecuencia, imputar, recoger más y/o nuevos datos.
- Análisis de los datos: Utilizando estadística descriptiva y técnicas de visualización puede ser de ayuda para que el científico de datos comprenda el contenido de los datos, evalúe la calidad y descubra patrones en los mismos. Una nueva visita de la etapa anterior, la recopilación de datos, podría ser necesario para completar adecuadamente esta fase.
- Preparación de los datos: La etapa de preparación de datos comprende todas las actividades que se utilizan para construir el conjunto de datos que separa la etapa de modelado. Estos incluyen la limpieza de datos, combinando datos de múltiples fuentes y transformación. La etapa de preparación de datos es el que más tiempo consume, aproximadamente 80 % del tiempo total del proyecto.

Sin embargo, se puede reducir el tiempo si los recursos de datos están bien gestionados, integrados y limpios. Automatizar algunos pasos de preparación de los datos puede reducir el tiempo aún más.

- Modelamiento: A partir de la primera versión del conjunto de datos preparados, la etapa de modelado se centra en el desarrollo de modelos predictivos o descriptivos de acuerdo al enfoque analítico previamente definido. Con los modelos predictivos, los científicos utilizan un conjunto de datos de entrenamiento (datos históricos en los que se conoce el resultado de interés o se tiene la columna clase) para construir el modelo . El proceso de modelado es típicamente muy iterativo. Para una determinada técnica , los científicos de datos pueden probar varios algoritmos con sus respectivos parámetros para encontrar el mejor modelo para las variables disponibles .

- Evaluación: Durante el desarrollo del modelo y antes de la implementación, el científico de datos lo evalúa y asegura que de forma apropiada y completa aborda el problema de negocio. La evaluación implica el cálculo de diversas medidas de diagnóstico, tablas y gráficos, que permiten al científico de datos interpretar la calidad del modelo y su eficacia en la solución del problema. Para un modelo predictivo, los científicos de datos utilizan un conjunto de pruebas, que es independiente del conjunto de entrenamiento, pero sigue la misma distribución de probabilidad y tiene un resultado conocido. El conjunto de prueba se utiliza para evaluar el modelo de lo que puede ser refinado según sea necesario.
- Despliegue: Después que el modelo es desarrollado y aprobado por los usuarios finales, se implementa un entorno de prueba. La implementación puede ser tan simple como la generación de un informe con recomendaciones, o tan complicado como la incorporación del modelo en un complejo proceso de flujo de trabajo.
- Retroalimentación: Mediante la recopilación de los resultados del modelo implementado, se recibe información sobre el desempeño. El análisis de esta información permite a los científicos de datos refinar el modelo para mejorar su precisión y utilidad. Se pueden automatizar algunos o todos los pasos de retroalimentación, recopilación y evaluación de modelo.

Adaptando esta metodología al presente trabajo especial de grado, los pasos quedan de la siguiente manera:

- Entender el negocio: se desea analizar redes que representen grandes volúmenes de datos y obtener valor, para poder realizar esta actividad es necesario extraer una red, representarla como un grafo, almacenarla, desarrollar operaciones, implementar algoritmos y utilizar herramientas de visualización. Para poder extraer una red se puede utilizar cualquier temática y fuente, puede ser almacenada en un sistema de archivos distribuido o en algún tipo de base de datos, es necesario tener un sistema de procesamiento de grafos y para lograr una representación gráfica hay que utilizar una herramienta especializada en visualización de redes.
- Enfoque analítico: Extraer las redes de la siguiente página <https://snap.stanford.edu/data/>. Para resolver el problema hay que definir una infraestructura de procesamiento distribuido y paralelo. Utilización de un clúster multi nodo basado en la distribución Hadoop, Cloudera. HDFS para almacenar las redes y Apache Spark para procesarlos. GraphX para implementar operaciones y algoritmos, estructurar las redes en grafos, contar la cantidad de nodos, aristas, grados de entrada, grados de salida, PageRank, Shortest Paths, Triangle Count y Connected Components. La herramienta Gephi para visualizar las redes.

- Requerimientos de los datos: Los datos deben representarse en un formato de listas de aristas. Los archivos sólo deben contener dos números enteros en cada línea, un identificador de origen y uno de destino.
- Recolección de los datos: Los datos fueron recolectados de la página <https://snap.stanford.edu/data/>.
- Análisis de los datos: Utilizar la herramienta Gephi para visualizar el grafo y así comprender el contenido de los datos.
- Preparación de los datos: Los datos están bien gestionados, integrados y limpios, por lo tanto no es necesario realizar un preprocesamiento.
- Modelamiento: En este proyecto no se desarrollarán modelos predictivos o descriptivos, por lo que esta fase no es necesaria.
- Evaluación: En este proyecto no se evaluarán modelos, por lo que esta fase no es necesaria.
- Despliegue: El entorno de prueba es un clúster multi nodo.
- Retroalimentación: Analizar los resultados de los algoritmos, en caso de ser necesario cambiar parámetros y volver a ejecutarlos.

CAPÍTULO 6

Marco aplicativo: Sandbox Cloudera

Se utilizó el Sandbox Cloudera para la implementación de los algoritmos en GraphX y realizar pruebas en un solo nodo. El Sandbox Cloudera provee herramientas del ecosistema Hadoop y sus respectivas configuraciones, por ejemplo, HDFS, Spark, GraphX, entre otros. Las características del equipo son las siguientes:

Componente	Especificación
Sistema Operativo	CentOS 6.7
Memoria	4 GiB
Procesador	Intel Core i5-3470 CPU @ 3.20GHz x 4
OS Type	64-bit
Disco	64 GB

Tabla 6.1: Descripción del equipo donde se encuentra instalado el Sandbox Cloudera.

6.1. Instalación de un solo nodo para pruebas (Sandbox Cloudera)

El primer paso es instalar una herramienta que proporcione un software de virtualización, en este proyecto se utilizó VMware Workstation Pro 12, el cual fue descargado del siguiente enlace https://my.vmware.com/web/vmware/info?slug=desktop_end_user_computing/vmware_workstation_pro/12_0. La instalación de VMware se desarrolló en el sistema operativo Windows 10.



Figura 6.1: Descarga e instalación de VMware Workstation Pro 12.

El siguiente paso fue descargar la versión QuickStart de la distribución Cloudera. El enlace es el siguiente <http://www.cloudera.com/downloads.html>.

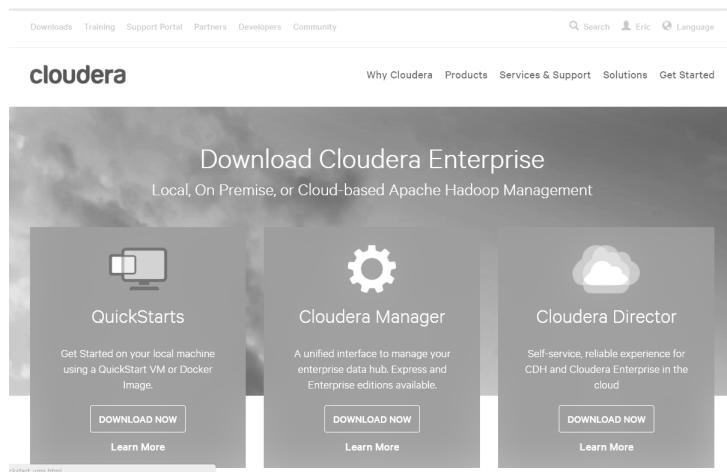


Figura 6.2: Descarga de Cloudera Quickstart VM CDH 5.7 Parte I.

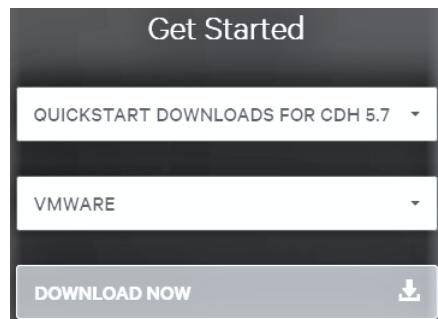


Figura 6.3: Descarga de Cloudera Quickstart VM CDH 5.7 Parte II.

Posteriormente se ejecuta VMware Workstation Pro 12 y se seleccionó la opción de Abrir una máquina virtual (Open a Virtual Machine).

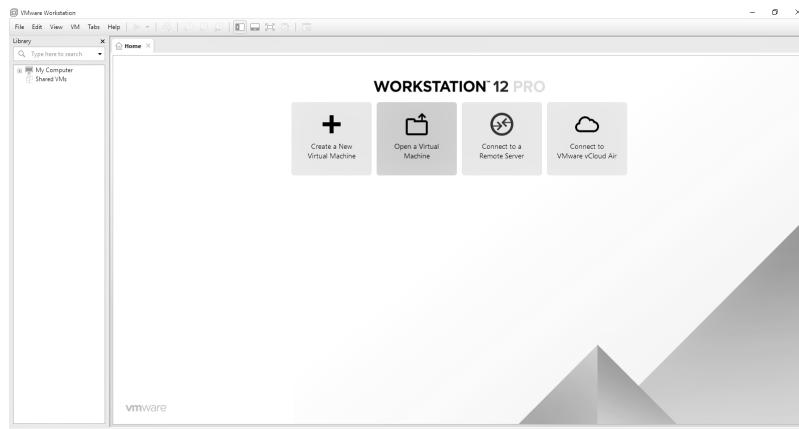


Figura 6.4: Abrir una máquina virtual.

Se utilizó la máquina virtual Cloudera Quickstart VM CDH 5.7, y finalmente se inició.

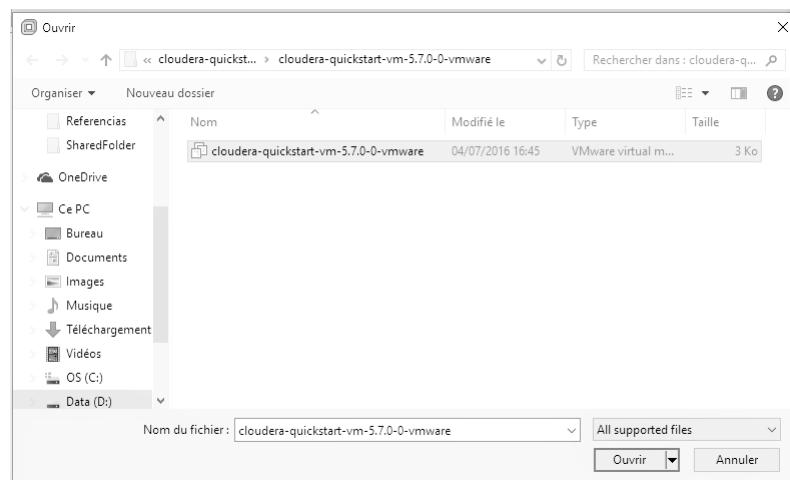


Figura 6.5: Seleccionar una máquina virtual.

6.2. Entorno

Se creó una carpeta con todos los componentes necesarios.

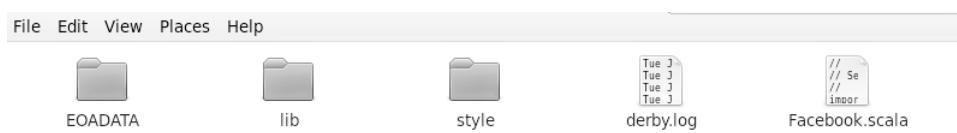


Figura 6.6: Conjunto de archivos utilizados.

En la carpeta EOADATA se encuentran todos los datasets y el script Facebook.scala se encuentra la implementación desarrollada para el análisis de un grafo que representa un subconjunto de la red social, Facebook. En la carpeta lib se encuentran todas las bibliotecas utilizadas.

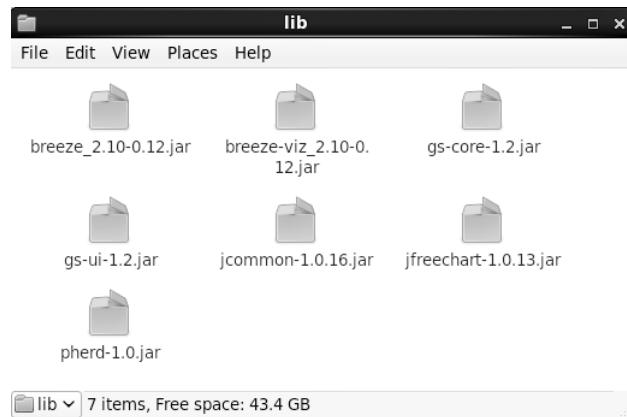


Figura 6.7: Bibliotecas utilizadas.

Breeze es un conjunto básico de bibliotecas para ScalaNLP, incluye algoritmos de álgebra lineal, cálculo numérico y optimización. Permite un enfoque genérico, potente y aún así eficiente para el aprendizaje automático y breeze-viz es un repositorio que permite visualizar diversas figuras.

6.3. Conjunto de datos de prueba (datasets)

En la carpeta EOADATA se encuentran todos los datasets utilizados para hacer pruebas.

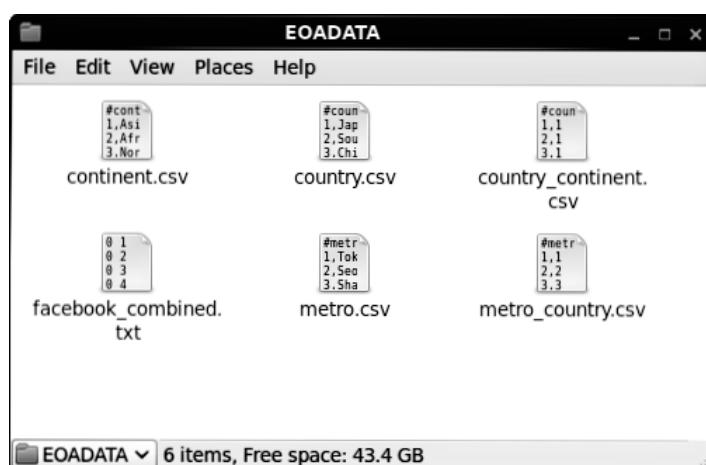


Figura 6.8: Datasets.

6.4. Implementación

En primer lugar se almacenan los datasets en HDFS.

```
[cloudera@quickstart ExamplesOfAnalytics]$ hdfs dfs -put EOADATA
```

Figura 6.9: Almacenar datos en HDFS.

```
[cloudera@quickstart ExamplesOfAnalytics]$ hdfs dfs -ls EOADATA
Found 6 items
-rw-r--r-- 1 cloudera cloudera      101 2016-06-21 07:42 EOADATA/continent.csv
-rw-r--r-- 1 cloudera cloudera     354 2016-06-21 07:42 EOADATA/country.csv
-rw-r--r-- 1 cloudera cloudera     156 2016-06-21 07:42 EOADATA/country_continent.csv
-rw-r--r-- 1 cloudera cloudera 854362 2016-06-21 07:42 EOADATA/facebook_combined.txt
-rw-r--r-- 1 cloudera cloudera    1402 2016-06-21 07:42 EOADATA.metro.csv
-rw-r--r-- 1 cloudera cloudera     430 2016-06-21 07:42 EOADATA.metro_country.csv
```

Figura 6.10: Visualizar datos en HDFS.

6.5. Pruebas

Se ejecuta la consola de Spark junto a las distintas bibliotecas y el script Facebook.scala.

```
[cloudera@quickstart ExamplesOfAnalytics]$ spark-shell --jars lib/gs-core-1.2.jar,lib/gs-ui-1.2.jar,lib/jcommon-0.16.jar,lib/jfreechart-1.0.13.jar,lib/breeze_2.10-0.12.jar,lib/breeze-viz_2.10-0.12.jar,lib/pherfd-1.0.jar -i Facebook.scala
```

Figura 6.11: Ejecución de Apache Spark y otros.

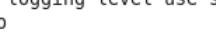
```
SLF4J: Class path contains multiple SLF4J bindings.  
SLF4J: Found binding in [jar:file:/usr/lib/zookeeper/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/  
StaticLoggerBinder.class]  
SLF4J: Found binding in [jar:file:/usr/jars/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerB  
inder.class]  
SLF4J: See http://www.slf4j.org/codes.html#multiple\_bindings for an explanation.  
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]  
Setting default log level to "WARN".  
To adjust logging level use sc.setLevel(newLevel).  
Welcome to  
  
version 1.6.0  
Using Scala version 2.10.5 (Java HotSpot(TM) 64-Bit Server VM, Java 1.7.0_67)  
Type in expressions to have them evaluated.  
Type :help for more information.
```

Figura 6.12: Ejecutando Spark Shell.

El script Facebook.scala permite importar el dataset, calcular la cantidad de vértices, aristas, grados de entrada y de salida.

Se importan los componentes a utilizar.

```
// Set log level to error, suppress info and warn messages
//
import org.apache.log4j.Logger
import org.apache.log4j.Level

Logger.getLogger("org").setLevel(Level.ERROR)
Logger.getLogger("akka").setLevel(Level.ERROR)

//
// Hands On: Building A Graph
//
import org.apache.spark.graphx._

val facebookGraph = GraphLoader.edgeListFile(sc, "./E0ADATA/facebook_combined.txt")

//
// Hands On: Network Connectedness and Clustering Components
//
import org.graphstream.graph.implementations._
```

Figura 6.13: Importando componentes.

Se realizan diversos cálculos y algoritmos.

```
val graph: SingleGraph = new SingleGraph("facebookGraph")

// Set up the visual attributes for graph visualization.
graph.addAttribute("ui.stylesheet", styleSheet);

graph.addAttribute("ui.quality")
graph.addAttribute("ui.antialias")

// Given the facebookGraph, load the graphX vertices into GraphStream
for ((id, _) <- facebookGraph.vertices.collect()) {
    graph.addNode(id.toString).asInstanceOf[SingleNode]
}

// Load the graphX edges into GraphStream edges
for ((Edge(x, y, _), count) <- facebookGraph.edges.collect().zipWithIndex) {
    graph.addEdge(count.toString, x.toString, y.toString).asInstanceOf[AbstractEdge]
}

// Display the graph.
graph.display()
```

Figura 6.14: Código implementado en Sandbox Cloudera.

6.6. Resultados

Finalmente se muestran los resultados de los cálculos y algoritmos realizados.

facebookGraph.numEdges
facebookGraph.numVertices

Figura 6.15: Código para calcular el número de vértices y aristas.

```

Loading Facebook.scala...
import org.apache.log4j.Logger
import org.apache.log4j.Level
import org.apache.spark.graphx_
facebookGraph: org.apache.spark.graphx.Graph[Int,Int] = org.apache.spark.graphx.
impl.GraphImpl@941c64
import org.graphstream.graph.implementations._
graph: org.graphstream.graph.implementations.SingleGraph = facebookGraph
res7: Long = 88234
res8: Long = 4039

```

Figura 6.16: Resultado número de vértices y aristas del dataset de Facebook.

```

def max(a: (VertexId, Int), b: (VertexId, Int)): (VertexId, Int) = {
  if (a._2 > b._2) a else b
}

def min(a: (VertexId, Int), b: (VertexId, Int)): (VertexId, Int) = {
  if (a._2 <= b._2) a else b
}

facebookGraph.outDegrees.reduce(max)
facebookGraph.inDegrees.reduce(max)

```

Figura 6.17: Algoritmos para calcular el vértice y su total de aristas que posea mayor InDegree y OutDegree.

```

res9: (org.apache.spark.graphx.VertexId, Int) = (107,1043)
res10: (org.apache.spark.graphx.VertexId, Int) = (1888,251)

```

scala> █

Figura 6.18: Resultados InDegree y OutDegree del dataset de Facebook.

Se muestra el grafo de forma visual utilizando la herramienta GraphStream.

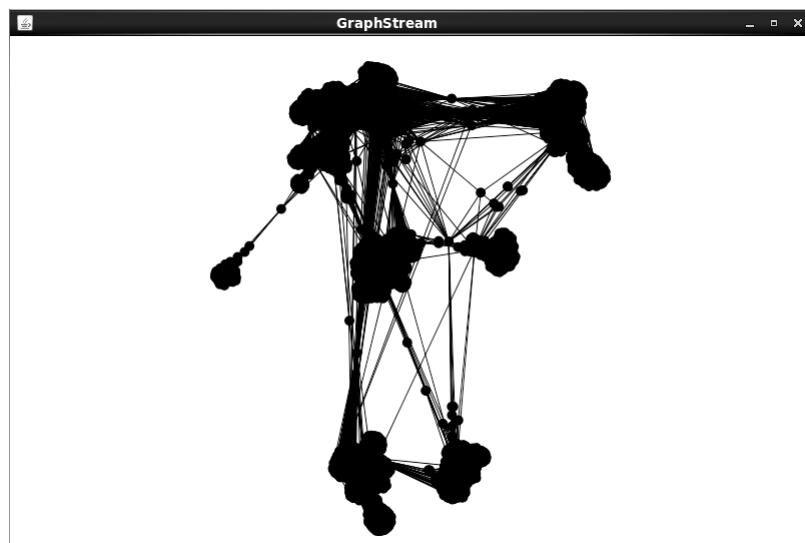


Figura 6.19: Visualización del dataset de Facebook utilizando GraphStream.

CAPÍTULO 7

Marco aplicativo: Clúster Hadoop

7.1. Clúster

Para el desarrollo de la solución se instaló un clúster Hadoop multi nodo (nodos físicos) utilizando la distribución Cloudera mediante 4 equipos conectados por LAN con las siguientes características:

Componente	Especificación
Sistema Operativo	CentOS 7
Memoria	3,7 GiB
Procesador	Intel Core i5-3470 CPU @ 3.20GHz x 4
OS Type	64-bit
Gráficos	Gallium 0.4 on NVD9
GNOME	Versión 3.14.2
Disco	980,1 GB

Tabla 7.1: Descripción de las máquinas utilizadas.

7.2. Instalación del sistema operativo CentOS 7

El sistema operativo que se instaló fue el CentOS 7 (Everything ISO) mediante un USB booteable. Las configuraciones de red y memoria fueron las siguientes:

Componente	Especificación
Ruta predeterminada:	190.169.70.254
DNS	190.169.94.5, 190.169.30.2
/boot	2048 MiB ext4
/	925,51 GiB ext4
/swap	4098 MiB

Tabla 7.2: Descripción de red y memoria.

7.3. Configuraciones de nodos

Para la creación del clúster se implementó un modelo maestro-esclavo, donde un nodo es el maestro y los otros tres los esclavos. A cada nodo se le asignó una dirección IPv4:

Nodo	Dirección IPv4
nodemaster	190.169.70.133
node1	190.169.70.136
node2	190.169.70.139
node3	190.169.70.123

Tabla 7.3: Dirección IPv4 de los nodos.

Para poder crear un clúster multi nodo donde la comunicación entre ellos sea posible y adecuada, es necesario realizar una serie de permisos, reglas de seguridad y configuraciones específicas en cada nodo.

7.3.1. Modificación del archivo host

Es necesario contener información persistente sobre los nombres de los equipos y direcciones IP de los nodos para poder desplegar el agente correctamente al instalar Cloudera Manager, por lo tanto hay que registrar las IPs de cada nodo en el fichero host de cada máquina de la siguiente manera. Esta configuración hay que realizarla en cada nodo.

```
$> nano /etc/hosts
127.0.0.1 localhost localhost.localdomain localhost4
localhost4.localdomain4
190.169.70.133 nodemaster
190.169.70.136 node1
190.169.70.139 node2
190.169.70.123 node3
::1 localhost localhost.localdomain
localhost6 localhost6.localdomain6
```

7.3.1.1. Creación de un hostname

Es necesario identificar cada máquina dentro de una red, por lo tanto se ejecuta el siguiente comando.

7.3.1.1.1. Nodo 1 (node1)

```
node1$> sudo hostname node1
```

7.3.1.1.2. Nodo 2 (node2)

```
node2$> sudo hostname node2
```

7.3.1.1.3. Nodo 3 (node3)

```
node3$> sudo hostname node3
```

7.3.1.1.4. Nodo Maestro (nodemaster)

```
nodemaster$> sudo hostname nodemaster
```

7.3.1.2. Configuración del archivo network

Se procede a especificar el hostname en el archivo network.

7.3.1.2.1. Nodo 1 (node1)

```
node1$> gedit /etc/sysconfig/network
NETWORKING=yes
HOSTNAME= node1
RES_OPTIONS="single-request-reopen"
```

7.3.1.2.2. Nodo 2 (node2)

```
node2$> gedit /etc/sysconfig/network
NETWORKING=yes
HOSTNAME= node2
RES_OPTIONS="single-request-reopen"
```

7.3.1.2.3. Nodo 3 (node3)

```
node3$> gedit /etc/sysconfig/network
NETWORKING=yes
HOSTNAME= node3
RES_OPTIONS="single-request-reopen"
```

7.3.1.2.4. Nodo Maestro (nodemaster)

```
nodemaster$> gedit /etc/sysconfig/network
NETWORKING=yes
HOSTNAME= nodemaster
RES_OPTIONS="single-request-reopen"
```

7.3.1.3. Desactivación del servicio SELINUX

SELinux es una extensión de seguridad de CentOS que debería proporcionar mayor seguridad. Debido a que este servicio por lo general causa más problemas (puede bloquear servicios o accesos) que ventajas y no es necesario mayor seguridad en este proyecto, podemos desactivarlo. Esta configuración hay que realizarla en cada nodo.

```
$> vim /etc/selinux/config
SELINUX = disable
```

7.3.1.4. Configuración del Network Time Protocol (NTP) y desactivación del Firewall

Network Time Protocol (NTP) es un protocolo de Internet para sincronizar los relojes de los sistemas informáticos a través del enrutamiento de paquetes en redes con latencia variable. Un firewall o cortafuegos es un dispositivo de hardware o un software que nos permite gestionar y filtrar la totalidad de tráfico entrante entre redes u ordenadores de una misma red. En resumen, es necesario que los nodos estén sincronizados y el Firewall este desactivado para que los nodos puedan comunicarse entre sí. Esta configuración hay que realizarla en cada nodo.

Configurar NTP.

```
$> sudo yum -y install ntp
$> sudo chkconfig ntpd on
$> sudo service ntpd start
$> sudo hwclock --systohc
```

Desactivar Firewall.

```
$> yum -y install iptables-services
$> service iptables stop && chkconfig iptables off
$> systemctl disable firewalld
```

7.3.1.5. Configuración del SSH

SSH (Secure SHell) es el nombre de un protocolo y del programa que lo implementa, y sirve para acceder a máquinas remotas a través de una red. Permite manejar por completo la computadora mediante un intérprete de comandos. SSH soporta múltiples formas para autenticar a los usuarios. Cloudera Manager deberá tener acceso SSH a los equipos del clúster para poder realizar la instalación y despliegue de servicios.

Modificación del archivo sshd_config en cada uno de los nodos.

```
$> nano /etc/ssh/sshd_config
Port 22
PermitRootLogin yes
RSAAuthentication yes
PubkeyAuthentication yes
$> service sshd restart
```

Generar las claves públicas sin password en cada uno de los nodos.

```
$> ssh-keygen -t rsa
```

Copiar todas las claves públicas en el nodo maestro.

```
nodemaster$> ssh-copy-id -i ~/.ssh/id_rsa.pub nodemaster
nodemaster$> ssh-copy-id -i ~/.ssh/id_rsa.pub node1
nodemaster$> ssh-copy-id -i ~/.ssh/id_rsa.pub node2
nodemaster$> ssh-copy-id -i ~/.ssh/id_rsa.pub node3
nodemaster$> chmod 0600 ~/.ssh/authorized_keys
nodemaster$> exit
nodemaster$> /etc/init.d/ssh restart
```

Finalmente se prueban las conexiones ssh, donde el nodo maestro puede acceder a los nodos esclavos sin clave:

```
$> su
$> ssh root@node1
$> logout
$> ssh root@node2
$> logout
$> ssh root@node3
$> logout
```

7.4. Instalación Cloudera Manager

Una vez finalizado las configuraciones anteriores se procede a la instalación de la herramienta Cloudera Manager en el nodo maestro (nodemaster):

```
$> wget http://archive.cloudera.com/cm5/installer/latest/
cloudera-manager-installer.bin
$> chmod u+x cloudera-manager-installer.bin
$> sudo ./cloudera-manager-installer.bin
```

Se utilizó Cloudera Enterprise Data Hub Edition Trial (también se puede utilizar Cloudera Express), y luego todo los parámetros por defecto.

Upgrading to **Cloudera Enterprise Data Hub Edition** provides important features that help you manage and monitor your Hadoop clusters in mission-critical environments.

Cloudera Express	Cloudera Enterprise Data Hub Edition Trial	Cloudera Enterprise	
License	Free	60 Days Alter the trial period, the product will continue to function as Cloudera Express . Your cluster and your data will remain unaffected.	Annual Subscription Upload License <input type="button" value="Select License File"/> <input type="button" value="Upload"/> Cloudera Enterprise is available in three editions: <ul style="list-style-type: none">• Basic Edition• Flex Edition• Data Hub Edition
Node Limit	Unlimited	Unlimited	Unlimited
CDH	✓	✓	✓
Core Cloudera Manager Features	✓	✓	✓
Advanced Cloudera Manager Features		✓	✓
Cloudera Navigator		✓	✓
Cloudera Navigator Key Trustee			✓
Cloudera Support			✓

Figura 7.1: Cloudera Enterprise Data Hub Edition Trial.

Se selecciona los hosts específicos para la instalación del clúster Cloudera de la siguiente forma:

190.169.70.136 , 190.169.70.139 , 190.169.70.133 , 190.169.70.123

Otra manera:

```
nodemaster
node1
node2
node3
```

Specify hosts for your CDH cluster installation.

New Hosts Currently Managed Hosts (1)

Hint: Search for hostnames and/or IP addresses using patterns [?P](#).

190.169.70.136, 190.169.70.139, 190.169.70.133, 190.169.70.123

SSH Port: 22

Figura 7.2: Selección de hosts para el CDH clúster.

Cloudera Manager realiza las instalaciones y configuraciones restantes en cada nodo.

Cluster Installation

Installation in progress.

Hostname	IP Address	Progress	Status	Details
node1	190.169.70.139	<div style="width: 50%;"> </div>	Installing jdk package...	Details
node2	190.169.70.133	<div style="width: 50%;"> </div>	Installing oracle-j2sdk1.7 package...	Details
node3	190.169.70.123	<div style="width: 50%;"> </div>	Installing oracle-j2sdk1.7 package...	Details
nodemaster	190.169.70.136	<div style="width: 100%; background-color: #2e7131;"> </div>	Installation completed successfully.	Details

Figura 7.3: Instalaciones y configuraciones para el clúster Cloudera.

Posteriormente se realiza la instalación de CDH en cada uno de los nodos de forma distribuida.

Cluster Installation

Installing Selected Parcels

The selected parcels are being downloaded and installed on all the hosts in the cluster.

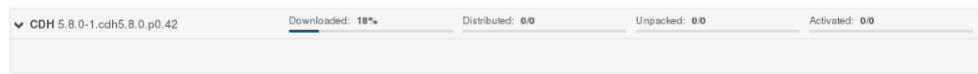


Figura 7.4: Instalación del CDH.

Se selecciona el core o las herramientas a utilizar, en este caso el core de Spark.

Cluster Setup

Choose the CDH 5 services that you want to install on your cluster.

Choose a combination of services to install.

- Core Hadoop**
HDFS, YARN (MapReduce 2 Included), ZooKeeper, Oozie, Hive, and Hue
- Core with HBase**
HDFS, YARN (MapReduce 2 Included), ZooKeeper, Oozie, Hive, and HBase
- Core with Impala**
HDFS, YARN (MapReduce 2 Included), ZooKeeper, Oozie, Hive, and Impala
- Core with Search**
HDFS, YARN (MapReduce 2 Included), ZooKeeper, Oozie, Hive, Hue, and Solr
- Core with Spark**
HDFS, YARN (MapReduce 2 Included), ZooKeeper, Oozie, Hive, Hue, and Spark
- All Services**
HDFS, YARN (MapReduce 2 Included), ZooKeeper, Oozie, Hive, Hue, HBase, Impala, Solr, Spark, and Key-Value Store Indexer
- Custom Services**
Choose your own services. Services required by chosen services will automatically be included. Flume can be added after your initial cluster has been set up.

This wizard will also install the **Cloudera Management Service**. These are a set of components that enable monitoring, reporting, events, and alerts; these components require databases to store information, which will be configured on the next page.

Figura 7.5: Selección del Core con Spark.

Configurar los usuarios, métodos de autenticación, contraseña, puerto SSH y el número de instalaciones simultáneas.

Instalación de clúster

Proporcionar credenciales de inicio de sesión de SSH.

Es necesario el acceso a raíz a los hosts para instalar los paquetes de Cloudera. Este instalador se conectará a los hosts mediante SSH e iniciará sesión directamente como raíz o como otro usuario con privilegios sudo/pbrun sin contraseña para convertirse en raíz.

Iniciar sesión en todos los hosts como: root Otro usuario

Puede realizar la conexión mediante autenticación por contraseña o clave pública para el usuario seleccionado anteriormente.

Método de autenticación: Todos los hosts aceptan la misma contraseña Todos los hosts aceptan la misma clave privada.

Introducir contraseña: (secreto)

Confirmar contraseña: (secreto)

Puerto SSH: 22

Número de instalaciones simultáneas: 4 (Ejecutar un gran número de instalaciones a la vez puede consumir una gran cantidad de ancho de banda y otros recursos del sistema)

Figura 7.6: Proporcionar credenciales de inicio de sesión de SSH.

Asignar los roles de HDFS para cada nodo.



Figura 7.7: Distribución de los roles para HDFS.

Asignar los roles de YARN y Spark para cada nodo.



Figura 7.8: Distribución de los roles para Spark y YARN.

Finalmente accedemos al Cloudera Manager mediante el localhost:7180.

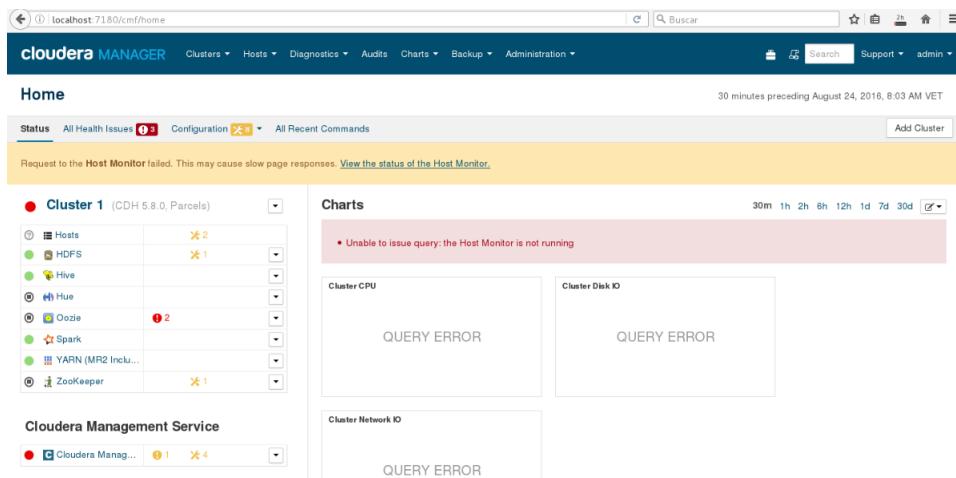


Figura 7.9: Cloudera Manager.

7.5. Instalación de SBT

Para este proyecto se utilizó una herramienta de construcción para proyectos llamada SBT, que funciona para implementaciones en Scala y Java (similar a Maven). Para poder ejecutar aplicaciones realizadas en los lenguajes mencionados anteriormente en Apache Spark mediante un clúster multi nodo utilizando YARN, es necesario crear una archivo JAR, por este motivo se creó un archivo build.sbt con todas las dependencias necesarias. Para instalar SBT es necesario ejecutar los siguientes comandos:

```
$> wget http://dl.bintray.com/sbt/rpm/sbt-0.13.5.rpm
$> sudo yum localinstall sbt-0.13.5.rpm
$> sbt -version
```

7.6. Extracción de datos

Se utilizaron diversos conjuntos de datos para aprender a utilizar GraphX, sin embargo para la finalidad de este trabajo especial de grado se utilizaron los siguientes:

Dataset	Descripción breve
Facebook	Representa usuarios y las relaciones entre ellos de la red social Facebook.
Egonets	El dataset Egonets representa múltiples egos del dataset Facebook.

Tabla 7.4: Datasets utilizados.

El dataset Facebook fue usado para las pruebas iniciales, debido a que no posee un gran número de nodos y aristas, fue ideal para utilizarlo en el Sandbox de Clou-

dera y para corroborar el funcionamiento del clúster. Fue descargado del siguiente enlace <https://snap.stanford.edu/data/facebook.tar.gz>.

El dataset Egonets representa múltiples egos del dataset Facebook. La unidad más pequeña de análisis en una red social es un individuo en su entorno social, es decir, un actor o ego. Una red tiene tantos egos como nodos tenga. Egos pueden ser personas, grupos, organizaciones o sociedades.

El análisis Egonetwork se centra en las características de la red, tales como el tamaño, la fuerza relación, densidad, centralidad, el prestigio y funciones tales como aislamientos, enlaces y puentes. Este tipo de análisis, se utilizan más comúnmente en los campos de la psicología o psicología social, el parentesco etnográfica análisis u otros estudios genealógicos de las relaciones entre los individuos. Fue descargado del siguiente enlace https://snap.stanford.edu/data/facebook_combined.txt.gz.

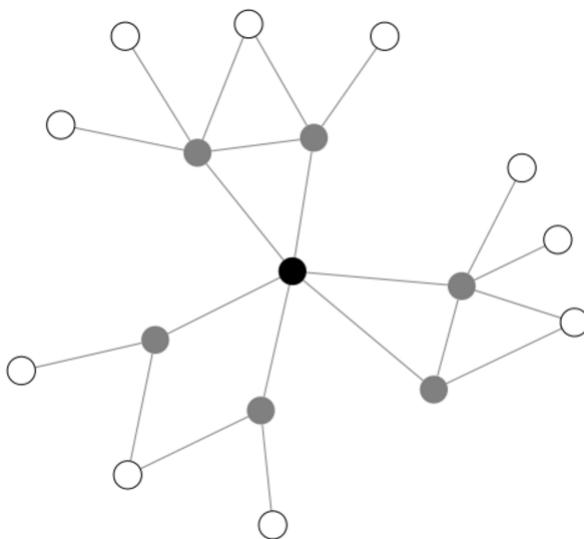


Figura 7.10: Ego network.

7.7. Implementación utilizando GraphX

A continuación se explicará las implementaciones realizadas de forma breve, todos los detalles y códigos completos se pueden encontrar en el repositorio GitHub ericbellet/Apache-Spark-GraphX.

7.7.1. Bibliotecas utilizadas

El primer paso es importar las bibliotecas a utilizar, Spark, GraphX y RDD:

```
import org.apache.spark.{SparkConf, SparkContext}
import org.apache.spark.graphx._
import org.apache.spark.rdd._
```

7.7.2. Configuraciones de Spark

Es necesario definir las propiedades de Spark para controlar la mayoría de los parámetros de la aplicación. Estas propiedades se pueden establecer directamente en la variable `SparkConf` para luego ser pasado al `SparkContext`. `SparkConf` permite configurar algunas de las propiedades comunes (por ejemplo, URL principal y nombre de la aplicación). Posterior a la definición de las propiedades se debe realizar la carga dinámica de estas en el `SparkContext`.

```
val sc = new SparkContext(
  new SparkConf().setAppName("Social_Media"))
```

7.7.3. Importar nodos y vértices

Para importar el grafo se puede utilizar el método `edgeListFile`, que se encuentra dentro del código `GraphLoader.scala` de la biblioteca `GraphX`. La función `edgeListFile` puede cargar un grafo desde un archivo que se posea un formato de listas de aristas.

```
376 1767
376 1777
376 1897
377 383
378 391
378 394
```

Figura 7.11: Ejemplo de formato de listas de aristas.

El archivo sólo debe contener dos números enteros en cada línea, un identificador de origen y uno de destino.

```
val grafo = GraphLoader.edgeListFile(sc, args(0))
```

7.7.4. Contar nodos y vértices

Si se desea saber cuantos nodos y vértices posee el grafo se utilizan los métodos `numVertices` para contar el número de vértices y `numEdges` para obtener el número de aristas.

```
val aristas = grafo.numEdges
val vertices = grafo.numVertices
```

7.7.5. Calcular grados de los nodos

Se definieron 2 funciones que permiten realizar un operación `reduce` para calcular cual es el grado máximo y mínimo de entrada y salida de todos los nodos.

```

def max(a: (VertexId , Int) ,
b: (VertexId , Int)): (VertexId , Int) = {
  if (a._2 > b._2) a else b
}

def min(a: (VertexId , Int) ,
b: (VertexId , Int)): (VertexId , Int) = {
  if (a._2 <= b._2) a else b
}

```

Para calcular el número de aristas que salen y entran de cada vértice se utilizan los métodos *indegree* para contar los que salen y *outdegree* los que entran.

```

grafo . outDegrees . reduce (max)
grafo . inDegrees . reduce (max)

grafo . outDegrees . reduce (min)
grafo . inDegrees . reduce (min)

```

7.7.6. Shortest Paths

El algoritmo de *ShortestPaths* proviene de la carpeta lib/ShortestPaths.scala de la biblioteca *org.apache.spark.graphx*. En lugar de calcular la trayectoria más corta en una sola fuente, calcula la distancia más corta entre cada dos vértices diferentes. El algoritmo ShortestPaths integrado en GraphX cuenta el número de saltos y devuelve la distancia en términos de número de saltos. El valor n representa todos los nodos de origen.

```
lib . ShortestPaths . run (grafo , Array (n))
```

Esta función retorna un arreglo donde cada posición posee 2 valores, el primer valor indica el vértice desde donde se inicia la trayectoria, y el segundo valor indica un Map(), donde el primer número es el nodo donde se termina la trayectoria y el segundo la cantidad de saltos, en el caso que el Map() no contenga valores significa que no existe una ruta.

7.7.7. Triangle Count

El algoritmo de *triangleCount* proviene de la carpeta lib/TriangleCount.scala de la biblioteca *org.apache.spark.graphx*.

```
lib . grafo . triangleCount ()
```

Esta función retorna un arreglo de tuplas, donde el primer valor de estas es el ID del vértice, y el segundo la cantidad de triángulos al cual pertenece.

7.7.8. PageRank

El algoritmo de *pageRank* proviene de la carpeta lib/PageRank.scala de la biblioteca *org.apache.spark.graphx*. El valor n pasado como parámetro es la tolerancia, que establece la compensación entre la velocidad y exactitud del resultado final. Si se coloca la tolerancia muy alta, el algoritmo se detendrá muy pronto y los resultados no serán tan precisos, y si es muy baja, el algoritmo continuará durante demasiado tiempo sin añadir nada a la precisión de los resultados.

```
grafo . pageRank( n )
```

Esta función retorna un arreglo de tuplas, donde el primer valor de estas es el ID del vértice, y el segundo un número real que representa su influencia en el grafo.

7.7.9. Connected Components

El algoritmo de *connectedComponents* proviene de la carpeta lib/connectedComponents.scala de la biblioteca *org.apache.spark.graphx*.

```
grafo . connectedComponents()
```

Esta función retorna un arreglo de CompactBuffer, donde cada uno contiene los vértices de cada componente.

7.7.10. Generar .gexf

La función *gexf* permite generar una extensión .gexf, la cual puede ser usada para realizar análisis sobre el grafo utilizando la herramienta Gephi.

```
def toGexf [VD,ED] ( g : Graph [VD,ED] ) =  
  "<?xml_version=\\\"1.0\\\" encoding=\\\"UTF-8\\\"?>\\n" +  
  "<gexf_xmlns=\\\"http://www.gexf.net/1.2/draft\\\"  
  version=\\\"1.2\\\">\\n" +  
  "  <graph_mode=\\\"static\\\" defaultedgetype=\\\"  
  directed\\\">\\n" +  
  "    <nodes>\\n" + g.vertices.map(v => "  
      <node_id=\\\"" + v._1 + "\\\"  
      label=\\\"" + v._2 + "\\\"/>\\n").collect.mkString + "  
    </nodes>\\n" + "    <edges>\\n" + g.edges.map(e => "  
      <edge_source=\\\"" + e.srcId + "\\\" target=\\\""  
      + e.dstId + "\\\" label=\\\"" +  
      e.attr + "\\\"/>\\n").collect.mkString  
    + "    </edges>\\n" +  
  "  </graph>\\n" + "</gexf>"
```

7.8. Almacenamiento de los conjuntos de datos en HDFS

En primer lugar hay que darle los permisos necesarios al usuario root u otro para poder utilizar HDFS en su plenitud.

```
$> sudo -u hdfs hadoop fs -mkdir /user/root
$> sudo -u hdfs hadoop fs -chown root /user/root
```

Para almacenar los datos en HDFS se creó una carpeta para cada dataset y posteriormente se almacenó.

Almacenar el dataset Facebook.txt.

```
$> hadoop fs -mkdir input
$> cd data
$> hadoop fs -put Facebook.txt input
```

Almacenar el dataset 1357.egonet.

```
$> hadoop fs -mkdir egonets
$> hadoop fs -put data/egonets/1357.egonet egonets
```

Comprobar que los datasets fueron almacenados correctamente.

```
$> hdfs dfs -ls
```

7.9. Extraer repositorio GitHub

Para este proyecto se utilizó el repositorio y manejador de versiones GitHub. El enlace del repositorio donde se encuentran las implementaciones, datos y configuraciones necesarias es el siguiente <https://github.com/ericbellet/Apache-Spark-GraphX>. El primer paso para extraer este repositorio es tener instalado git.

```
$> yum install git
$> git --version
```

Finalmente se extrae el repositorio y se instalan las dependencias utilizando la herramienta para la construcción de proyectos, SBT.

```
$> git clone \
https://github.com/ericbellet/Apache-Spark-GraphX.git
$> cd Apache-Spark-GraphX
$> sbt package
```

7.10. Pruebas

Una vez clonado el repositorio, se procede a acceder a él y a ejecutar el comando `spark-submit` en modo YARN cluster. El comando `spark-submit` provee una secuencia de comandos para iniciar aplicaciones en un clúster. Para poder utilizar los algoritmos en un ambiente distribuido y paralelo en Spark mediante YARN, es necesario poseer un archivo .jar que contenga las implementaciones realizadas.

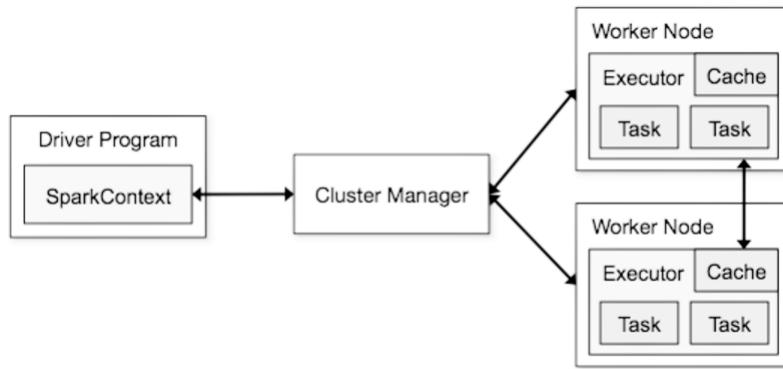


Figura 7.12: Spark modo clúster.

Para realizar las pruebas se crearon 6 módulos, donde cada uno fue ejecutado de manera independiente en modo YARN cluster y YARN client. Todos los módulos son particiones del código implementado, a excepción del módulo *Descripcion* que representa la totalidad de la implementación.

Módulos	Resumen
ShortestPaths	Calcula todas las distancias entre todos los nodos.
PageRank	Mide la influencia de los vértices de un grafo.
TriangleCount	Mide que tan conectado se encuentra un grafo.
ConnectedComponents	Calcula las componentes de una egonet.
Gephi	Genera un .gexf que puede ser utilizado en Gephi.
Descripcion	Calcula la cantidad de nodos y aristas del grafo, los grados de entrada y salida de todos los nodos, máximo grado de entrada y salida, mínimo grado de entrada y salida, la distancia promedio entre todos los nodos y la cantidad de triángulos totales que posee el grafo.

Tabla 7.5: Módulos de pruebas.

Spark utiliza una arquitectura maestro-esclavo, posee un coordinador central (Driver) que se comunica con trabajadores (workers) distribuidos, llamados ejecutores (executors). El coordinador central es el proceso en el que el método principal se ejecuta. Se encarga de dividir las aplicaciones en tareas y asignárselas a los ejecutores. Los ejecutores permiten la ejecución de tareas individuales en cada nodo, y los resultados son enviados al coordinador principal. El comando que fue utilizado para ejecutar la aplicación en el clúster, fue el siguiente:

```
$> spark-submit \
--class com.cloudera.sparksocialmedia.SparkSocialMedia \
--master yarn --deploy-mode cluster \
--num-executors 4 --driver-memory 2g \
--executor-memory 2g --executor-cores 4 \
target/scala-2.10/grafos-de-gran-escala_2.10-1.0.jar
input egonets \
Descripcion ShortestPaths \
TriangleCount PageRank ConnectedComponents
```

El parámetro *-class* es el punto de entrada para la aplicación, *-master* la dirección URL principal para el clúster, en este caso YARN en modo clúster *-deploy-mode cluster*, ya que se desea que los trabajos se desplieguen por los nodos.

La cantidad de ejecutores para la aplicación (*-num-executors*), en este caso se utilizaron 4, un ejecutor por cada nodo, la memoria del utilizada del controlador principal fue de 2g (*-driver-memory*), la memoria de los ejecutores fue de 2g (*-executor-memory*) y 4 cores para cada ejecutor (*-executor-cores*).

El resto de los parámetros son, el archivo .jar, los archivos de entrada de HDFS para la aplicación, en este caso *input* y *egonets* y los de salida, *Descripcion*, *ShortestPaths*, *TriangleCount*, *PageRank* y *ConnectedComponents*.

Si se desea ejecutar el código nuevamente es necesario eliminar los archivos en HDFS que contienen los resultados.

```
$> hdfs dfs -rmr Descripcion
$> hdfs dfs -rmr ShortestPaths
$> hdfs dfs -rmr TriangleCount
$> hdfs dfs -rmr PageRank
$> hdfs dfs -rmr ConnectedComponents
```

Luego de realizar las pruebas en el clúster se decidió ejecutar los algoritmos en el SandBox Cloudera, para poder comparar los tiempos de ejecución. Debido a que el SandBox Cloudera especificado en el capítulo anterior (Capítulo 6) posee un solo nodo, se ejecutó en modo YARN client, donde este obtiene el rol de ResourceManager y ejecutor. El nodo utilizado para el SandBox Cloudera es similar a un solo nodo del clúster.

```
$> spark-submit \
--class com.cloudera.sparksocialmedia.SparkSocialMedia \
--master yarn --deploy-mode client \
target/scala-2.10/grafos-de-gran-escala_2.10-1.0.jar \
input egonets \
Descripcion ShortestPaths \
TriangleCount PageRank ConnectedComponents
```

7.11. Resultados

Los resultados almacenados en HDFS pueden ser visualizados en consola mediante el comando *-cat*, y pueden ser extraídos a una carpeta local en una extensión .txt, mediante el comando nativo de Hadoop y HDFS, *-get*. Spark permite la visualización de los logs de ejecución mediante una interfaz web llamada *HistoryServer*, que indica diversos detalles del procesamiento de la aplicación, como el tiempo total, fecha de inicio, fecha de finalización, tiempo de cada proceso, memoria utilizada, entre otros. YARN tambien posee una interfaz de usuario llamada *ResourceManager UI* que también provee información de la ejecución de la aplicación.

Todos los resultados fueron transferidos de HDFS a una carpeta local con el nombre *output*, en el caso del resultado de la función *toGexf*, es decir, el archivo *grafo.gexf* se almacenó en la carpeta *gephi*.

```
$> hadoop fs -cat Descripcion/*
$> hadoop fs -get Descripcion ./output

$> hadoop fs -cat ShortestPaths/*
$> hadoop fs -get ShortestPaths ./output

$> hadoop fs -cat TriangleCount/*
$> hadoop fs -get TriangleCount ./output

$> hadoop fs -cat PageRank/*
$> hadoop fs -get PageRank ./output

$> hadoop fs -cat ConnectedComponents/*
$> hadoop fs -get ConnectedComponents ./output

$> cd Apache-Spark-GraphX/gephi
```

Para monitorear los procesos en el clúster se puede utilizar tanto el *HistoryServer* como el *ResourceManager UI*, sin embargo en el SandBox de Cloudera solo se puede utilizar el *ResourceManager UI*. Cada uno de los módulos fue ejecutado individualmente en el clúster y en el Sandbox de Cloudera, y se obtuvieron los siguientes resultados.

7.11.1. ShortestPaths

El módulo *ShortestPaths* tardó 48 minutos y realizó 22 trabajos (jobs) en el modo YARN clúster. En modo YARN client tardó 2 horas con 7 minutos y 7 segundos.

```
(3556 Map) 8972 > 4, 2962 > 4, 3844 > 5, 3889 > 4, 2938 > 5, 3939 > 5, 3838 > 2, 3839 > 2, 3838 > 2, 3834 > 2, 3812 > 2, 3777 > 2, 3947 > 1, 3738 > 4, 3898 >
4, 3880 > 4, 3878 > 3, 3891 > 5, 3797 > 2, 3938 > 4, 3821 > 2, 3617 > 2, 3845 > 4, 3819 > 2, 3855 > 2, 3869 > 2, 3861 > 5, 3856 > 2, 3762 > 2, 3777 > 2, 3947 > 1, 3738 > 4, 3898 >
3870 > 1, 3905 > 1, 3899 > 3, 3852 > 4, 3635 > 4, 3767 > 3, 3795 > 6, 3711 > 1, 3972 > 2, 3752 > 2, 3884 > 5, 3816 > 2, 3893 > 4, 3948 > 4,
3958 > 1, 3818 > 1, 3795 > 5, 3718 > 2, 3877 > 4, 3926 > 4, 3763 > 2, 3667 > 5, 3931 > 2, 3795 > 3, 3731 > 4, 3964 > 2, 3784 > 4, 3989 > 1,
3707 > 2, 3769 > 4, 3860 > 5, 3861 > 4, 3862 > 5, 3943 > 1, 3838 > 2, 3841 > 4, 3707 > 5, 3889 > 2, 3738 > 1, 3888 > 5, 3839 > 5,
3937 > 2, 3769 > 5, 3967 > 3, 3892 > 5, 3824 > 5, 3943 > 1, 3838 > 2, 3841 > 4, 3707 > 5, 3889 > 2, 3738 > 1, 3888 > 5, 3839 > 5,
3792 > 1, 3868 > 5, 3741 > 5, 3768 > 1, 3736 > 2, 3758 > 1, 3945 > 2, 3698 > 2, 3664 > 3, 3795 > 5, 3864 > 3, 3646 > 4, 3951 > 5, 3832 > 4,
3846 > 4, 3867 > 4, 3674 > 4, 3953 > 6, 3786 > 2, 3986 > 5, 3881 > 1, 3672 > 4, 3975 > 3, 3737 > 4, 3640 > 4, 3913 > 4, 3842 > 4, 3984 > 2,
3754 > 2, 3833 > 4, 2959 > 6, 3798 > 1, 3872 > 2, 3884 > 5, 3810 > 2, 3823 > 3, 3828 > 5, 3852 > 5, 3865 > 5, 3866 > 5, 3835 > 5,
3920 > 2, 3864 > 4, 3865 > 5, 3866 > 4, 3867 > 5, 3868 > 1, 3869 > 2, 3870 > 5, 3871 > 4, 3872 > 5, 3873 > 4, 3911 > 4,
3864 > 4, 3825 > 2, 3859 > 2, 3798 > 6, 3978 > 3, 3844 > 4, 3845 > 2, 3877 > 4, 3984 > 4, 3633 > 2, 3761 > 2, 3655 > 4, 3869 > 4, 3785 > 5,
3865 > 5, 3886 > 4, 3918 > 3, 3969 > 4, 3958 > 5, 3924 > 3, 3933 > 3, 3721 > 40)
```

Figura 7.13: Vista previa del resultado de ShortestPaths.

The screenshot shows the 'Completed Jobs (22)' section of the Cloudera Manager Spark Jobs UI. It lists 22 completed jobs with details like Job ID, Description, Submitted time, Duration, Stages, and Tasks. Most tasks were successful, with some skipped due to being part of a reduce stage.

Job ID	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
21	ssaveTxtFile at SparkSocialMedia scala_105	2016/08/29 16:22:28	1/1		0/1 (0 skipped)
20	collect at SparkSocialMedia scala_105	2016/08/29 16:22:14	12 s	1/1 (477 skipped)	22/254 (8 skipped)
19	reduce at VertexRDDImpl scala_90	2016/08/29 16:22:13	0.6 s	4/4 (530 skipped)	8/8 (158 skipped)
18	reduce at VertexRDDImpl scala_90	2016/08/29 16:22:11	2 s	4/4 (472 skipped)	8/8 (148 skipped)
17	reduce at VertexRDDImpl scala_90	2016/08/29 16:22:07	4 s	4/4 (452 skipped)	8/8 (144 skipped)
16	reduce at VertexRDDImpl scala_90	2016/08/29 16:21:43	23 s	4/4 (371 skipped)	8/8 (138 skipped)
15	reduce at VertexRDDImpl scala_90	2016/08/29 16:21:21	22 s	4/4 (327 skipped)	8/8 (104 skipped)
14	reduce at VertexRDDImpl scala_90	2016/08/29 16:21:03	18 s	4/4 (281 skipped)	8/8 (88 skipped)
13	reduce at VertexRDDImpl scala_90	2016/08/29 16:19:47	1.3 min	4/4 (244 skipped)	8/8 (69 skipped)
12	reduce at VertexRDDImpl scala_90	2016/08/29 16:17:23	2.4 min	4/4 (207 skipped)	8/8 (54 skipped)
11	reduce at VertexRDDImpl scala_90	2016/08/29 16:11:11	6.2 min	4/4 (173 skipped)	8/8 (49 skipped)
10	reduce at VertexRDDImpl scala_90	2016/08/29 16:09:39	2.5 min	4/4 (142 skipped)	8/8 (28 skipped)
9	reduce at VertexRDDImpl scala_90	2016/08/29 15:57:25	11 min	4/4 (114 skipped)	8/8 (228 skipped)
8	reduce at VertexRDDImpl scala_90	2016/08/29 15:54:49	2.6 min	4/4 (89 skipped)	8/8 (178 skipped)
7	reduce at VertexRDDImpl scala_90	2016/08/29 15:45:26	0.4 min	4/4 (67 skipped)	8/8 (134 skipped)

Figura 7.14: Información de ejecución del módulo ShortestPaths en modo YARN clúster.

The screenshot shows the 'Application Overview' section of the Cloudera Manager Application Overview UI. It displays details for the 'Social Media' application, including User (root), Application Type (SPARK), Application Tags, State (FINISHED), FinalStatus (SUCCEEDED), Start date (Sun Sep 04 10:38:07 -0700 2016), End date (2 hrs, 7 mins, 7 sec), Tracking URL (History), and Diagnostics.

User:	root
Name:	Social Media
Application Type:	SPARK
Application Tags:	
State:	FINISHED
FinalStatus:	SUCCEEDED
Started:	Sun Sep 04 10:38:07 -0700 2016
Elapsed:	2 hrs, 7 mins, 7 sec
Tracking URL:	History
Diagnostics:	

Figura 7.15: Información de ejecución del módulo ShortestPath en modo YARN client.

7.11.2. PageRank

El módulo *PageRank* tardó 40 segundos y realizó 19 trabajos (jobs) en el modo YARN clúster. En modo YARN client tardó 2 minutos con 43 segundos.

part=00000 [Sólo lectura]	
Abrir	X
(1911, 18, 2089718694691)	
(3434, 18, 149538262241884)	
(2649, 18, 129278531521985)	
(1982, 17, 359718947869787)	
(1888, 13, 312179928536056)	
(2649, 12, 129278531521985)	
(1987, 13, 348593936318129)	
(391, 9, 53169822275996)	
(2654, 9, 53169822275996)	
(1910, 8, 116776942465712)	
(1894, 7, 67956865459837)	
(2650, 7, 222249634242345)	
(1898, 7, 29365272129942)	
(1882, 7, 694457338956258)	
(3426, 7, 119529736230187)	
(2642, 5, 956254915766993)	
(322, 5, 71214349906337)	
(3422, 5, 707918720395754)	
(1881, 5, 654938176954023)	
(2652, 5, 222249634242345)	
(3968, 5, 222249634242345)	
(1986, 5, 111531053613266)	
(1897, 5, 105956519882791)	
(1883, 5, 2645882578989856)	
(2638, 4, 533082508130165)	
(2643, 4, 658865301234411)	
(853, 4, 76487461259334)	
(2639, 4, 76487461259334)	
(1880, 4, 213243496158395)	
(1884, 4, 581675463933338)	
(2646, 4, 575481185123768)	
(2629, 4, 772346727299)	
(3431, 4, 5426882578989856)	
(3496, 4, 2668825789898563)	
(3397, 4, 262458368654482)	
(1983, 4, 250696803172977)	
(2637, 4, 250696803172977)	
(3419, 4, 1423687311564095)	
(2631, 4, 6254469158194475)	
(1864, 3, 9703790563938432)	
(3948, 3, 8519728198478177)	

Figura 7.16: Vista previa del resultado de PageRank.

Spark - Cloudera Manager - Social Media - Spark Jobs						
nodo1:18088/history/application_1472496755488_0004/1/jobs/		Spark Jobs (?)				
Total Uptime: 40 s						Scheduling Mode: FIFO
Completed Jobs: 19						► Event Timeline
Completed Jobs (19)						
Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total	
18	saveAsTextFile at SparkSocialMedia.scala:121	2016/08/29 15:23:29	0.8 s	1/1	1/1	
17	collect at SparkSocialMedia.scala:121	2016/08/29 15:23:29	0.4 s	2/2 (359 skipped)	4/4 (718 skipped)	
16	sortBy at SparkSocialMedia.scala:121	2016/08/29 15:23:28	63 ms	1/1 (359 skipped)	2/2 (718 skipped)	
15	reduce at VertexRDDImpl.scala:90	2016/08/29 15:23:28	0.4 s	4/4 (404 skipped)	8/8 (808 skipped)	
14	reduce at VertexRDDImpl.scala:90	2016/08/29 15:23:28	0.3 s	4/4 (356 skipped)	8/8 (712 skipped)	
13	reduce at VertexRDDImpl.scala:90	2016/08/29 15:23:27	0.2 s	4/4 (311 skipped)	8/8 (622 skipped)	
12	reduce at VertexRDDImpl.scala:90	2016/08/29 15:23:27	0.3 s	4/4 (289 skipped)	8/8 (558 skipped)	
11	reduce at VertexRDDImpl.scala:90	2016/08/29 15:23:27	0.3 s	4/4 (230 skipped)	8/8 (480 skipped)	
10	reduce at VertexRDDImpl.scala:90	2016/08/29 15:23:26	0.2 s	4/4 (194 skipped)	8/8 (398 skipped)	
9	reduce at VertexRDDImpl.scala:90	2016/08/29 15:23:26	0.2 s	4/4 (181 skipped)	8/8 (322 skipped)	
8	reduce at VertexRDDImpl.scala:90	2016/08/29 15:23:26	0.3 s	4/4 (131 skipped)	8/8 (262 skipped)	
7	reduce at VertexRDDImpl.scala:90	2016/08/29 15:23:26	0.2 s	4/4 (104 skipped)	8/8 (208 skipped)	
6	reduce at VertexRDDImpl.scala:90	2016/08/29 15:23:25	0.2 s	4/4 (80 skipped)	8/8 (160 skipped)	
5	reduce at VertexRDDImpl.scala:90	2016/08/29 15:23:25	0.3 s	4/4 (59 skipped)	8/8 (118 skipped)	
4	reduce at VertexRDDImpl.scala:90	2016/08/29 15:23:25	0.2 s	4/4 (41 skipped)	8/8 (82 skipped)	
3	reduce at VertexRDDImpl.scala:90	2016/08/29 15:23:25	0.2 s	4/4 (26 skipped)	8/8 (52 skipped)	
2	reduce at VertexRDDImpl.scala:90	2016/08/29 15:23:24	0.3 s	4/4 (14 skipped)	8/8 (28 skipped)	
1	reduce at VertexRDDImpl.scala:90	2016/08/29 15:23:23	1.0 s	7/7 (2 skipped)	14/14 (4 skipped)	

Figura 7.17: Información de ejecución del módulo PageRank en modo YARN cluster.

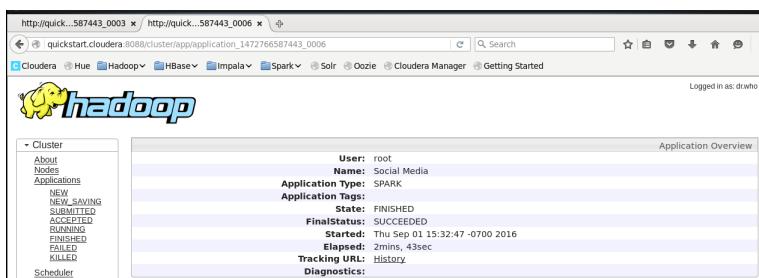


Figura 7.18: Información de ejecución del módulo PageRank en modo YARN client.

7.11.3. TriangleCount

El módulo *TriangleCount* tardó 37 segundos y realizó 3 trabajos (jobs) en el modo YARN clúster. En modo YARN client tardó 3 minutos con 12 segundos.

part-00000 [Sólo lectura]	
<small>-r/Escritorio/Apache-Spark-GraphX/output/TriangleCount</small>	
(3558,287)	
(107,73)	
(1410,145)	
(3456,692)	
(3762,374)	
(3272,394)	
(3068,55)	
(4938,29)	
(1898,1951)	
(1949,1416)	
(467,44)	
(912,17)	
(2958,219)	
(149,49)	
(363,95)	
(556,37)	
(1466,0)	
(3766,188)	
(2334,7139)	
(42,12)	
(2596,6129)	
(456,148)	
(492,1196)	
(160,7)	
(1148,1193)	
(1168,8)	
(1596,1793)	
(1738,77)	
(1824,49)	
(2346,451)	
(2594,2229)	
(3932,25)	
(297,734)	
(3659,6)	
(1580,217)	
(3336,21)	
(467,867)	
(2660,1888)	
(1399,8899)	
(548,224)	
(1630,760)	
(1732,741)	

Figura 7.19: Vista previa del resultado de TriangleCount.

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
2	saveAsTextFile at SparkSocialMedia.scala:115	2016/08/30 09:43:54	0,8 s	1/1	1/1
1	collect at SparkSocialMedia.scala:115	2016/08/30 09:43:52	2 s	5/5	10/10
0	count at GraphLoader.scala:93	2016/08/30 09:43:49	2 s	1/1	2/2

Figura 7.20: Información de ejecución del módulo TriangleCount en modo YARN cluster.

User:	root
Name:	Social Media
Application Type:	SPARK
Application Tags:	
State:	FINISHED
Final Status:	SUCCEEDED
Started:	Fri Sep 02 08:52:44 -0700 2016
Elapsed:	3mins, 12sec
Tracking URL:	History
Diagnostics:	

Figura 7.21: Información de ejecución del módulo TriangleCount en modo YARN client.

7.11.4. ConnectedComponents

El módulo *ConnectedComponents* tardó 5 minutos con 6 segundos y realizó 864 trabajos (jobs) en el modo YARN clúster. En modo YARN client tardó 17 minutos con 15 segundos.

Figura 7.22: Vista previa del resultado de ConnectedComponents.

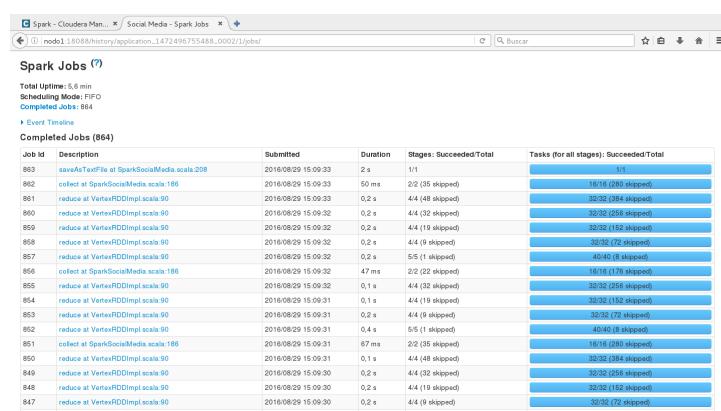


Figura 7.23: Información de ejecución del módulo ConnectedComponents en modo YARN cluster.

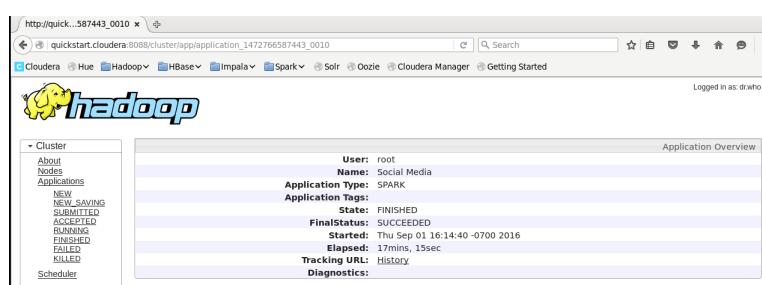


Figura 7.24: Información de ejecución del módulo ConnectedComponents en modo YARN client.

7.11.5. Gephi

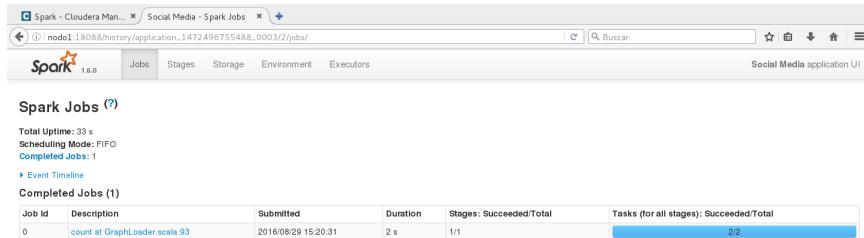
El módulo *Gephi* tardó 33 segundos y realizó 1 trabajo (jobs) en el modo YARN clúster. En modo YARN client tardó 1 minutos con 25 segundos.



```
<?xml version="1.0" encoding="UTF-8"?>
<gefx xmlns="http://www.gephi.org/1.2draft" version="1.2">
<graph mode="static" defaultedgetype="directed">
<nodes>
<node id="384" label="1" />
<node id="1004" label="1" />
<node id="3792" label="1" />
<node id="3897" label="1" />
<node id="667" label="1" />
<node id="323" label="1" />
<node id="1894" label="1" />
<node id="2493" label="1" />
<node id="1325" label="1" />
<node id="3517" label="1" />
<node id="2577" label="1" />
<node id="149" label="1" />
<node id="204" label="1" />
<node id="956" label="1" />
<node id="1000" label="1" />
<node id="1" label="1" />
<node id="3899" label="1" />
<node id="3973" label="1" />
<node id="2232" label="1" />
<node id="2234" label="1" />
<node id="755" label="1" />
<node id="1813" label="1" />
<node id="2326" label="1" />
<node id="2349" label="1" />
<node id="1393" label="1" />
<node id="456" label="1" />
<node id="2021" label="1" />
<node id="1596" label="1" />
<node id="1789" label="1" />
<node id="3165" label="1" />
<node id="2345" label="1" />
<node id="2346" label="1" />
<node id="3932" label="1" />
<node id="2117" label="1" />
<node id="2908" label="1" />
<node id="255" label="1" />
<node id="1569" label="1" />

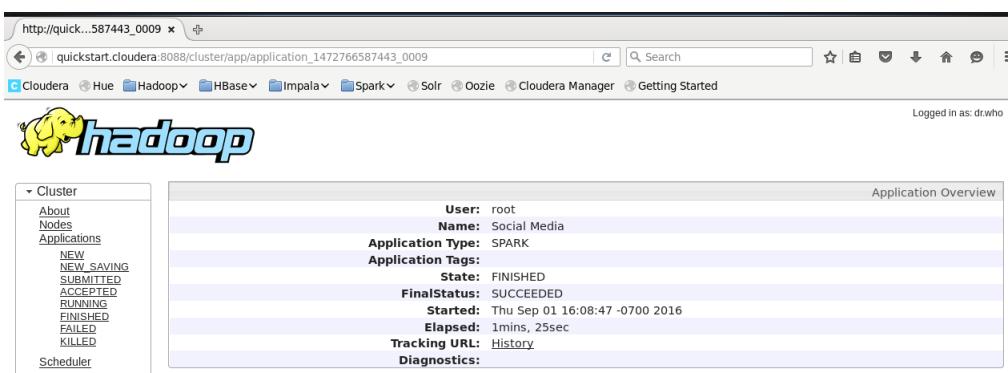
```

Figura 7.25: Vista previa del resultado de Gephi.



Job ID	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
0	count at GraphLoader.scala:93	2018/08/29 15:20:31	2 s	1/1	2/2

Figura 7.26: Información de ejecución del módulo Gephi en modo YARN cluster.



User:	root
Name:	Social Media
Application Type:	SPARK
Application Tags:	
State:	FINISHED
FinalStatus:	SUCCEEDED
Started:	Thu Sep 01 16:08:47 -0700 2016
Elapsed:	1mins, 25sec
Tracking URL:	History
Diagnostics:	

Figura 7.27: Información de ejecución del módulo Gephi en modo YARN client.

7.11.6. Descripcion

El módulo *Descripcion* tardó 50 segundos y realizó 44 trabajos (jobs) en el modo YARN clúster. En modo YARN client tardó 5 minutos con 15 segundos.

```

1 Número de vértices: 4039.
2 Número de aristas: 88234.
3 Máximo outDegrees: Nodo -> 107, Grados -> 1043.
4 Máximo inDegrees: Nodo -> 1888, Grados -> 251.
5 Mínimo outDegrees: Nodo -> 1894, Grados -> 1.
6 Mínimo inDegrees: Nodo -> 956, Grados -> 1.
7 Total de triángulos: 4836030.
8 Usuarios más influyentes:
9 (1911,18.20897718694691), (3434,18.148638262241864),
10 (2655,17.529806380070582), (1902,17.359718047069787),
11 (1888,13.312179928536056), (2649,12.129278531521985),
12 (1907,9.948390834318129), (3971,9.803693938215641),
13 (2654,9.531068222775996), (1910,8.116776942405712).
14 Distancia promedio entre todos los nodos del grafo: 4.

```

Figura 7.28: Vista previa del resultado de Descripcion.

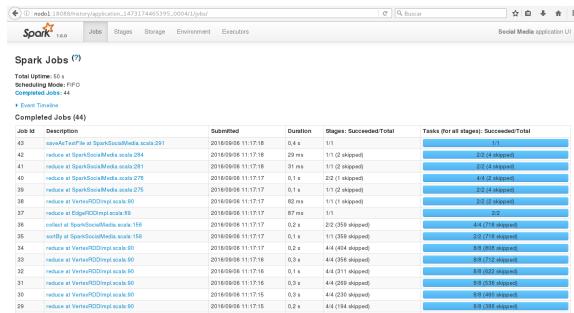


Figura 7.29: Información de ejecución del módulo Descripcion en modo YARN clúster.



Figura 7.30: Información de ejecución del módulo Descripcion en modo YARN client.

Podemos observar en los resultados que el grafo posee 4039 vértices, 88234 aristas, el nodo con mayor grado de salida es el *107* con un valor de 1043, con mayor grado de entrada es el *1888* con un valor de 251. El usuario con mayor influencia es el *1911*. La distancia promedio entre todos los nodos del grafo es de 4 saltos, y el total de triángulos es de 4836030.

CAPÍTULO 8

Marco aplicativo: Gephi

Para visualizar los grafos se utilizó la herramienta Gephi, debido a que es una herramienta de análisis de redes de fácil acceso y poderosa. El análisis y visualización de la red es una técnica interesante para dar al investigador la posibilidad de ver sus datos desde un nuevo ángulo.

8.1. Importar grafo

El primer paso es importar el grafo a la herramienta, Gephi soporta diversos formatos, en este caso se utilizó la extensión .gexf.

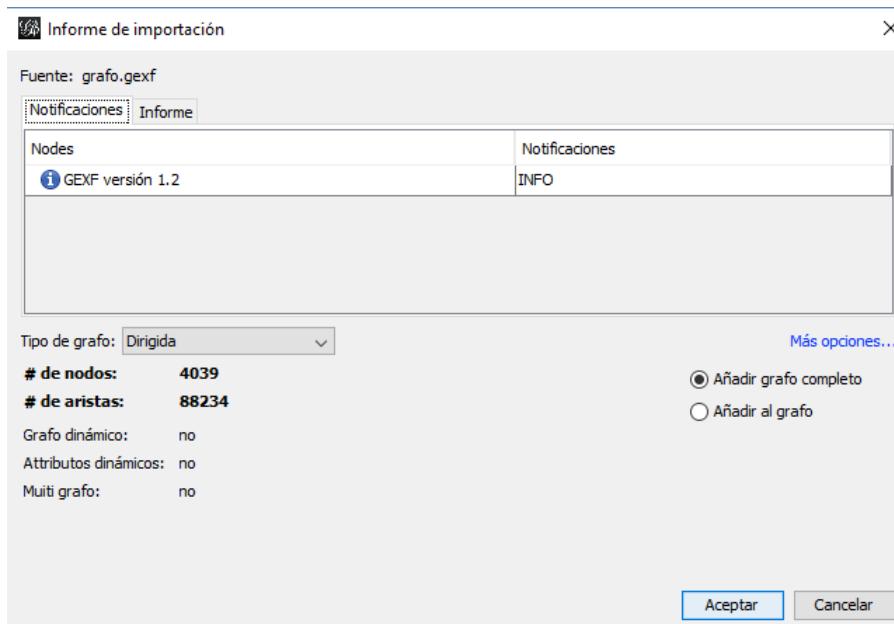


Figura 8.1: Importar grafo a Gephi.

Inmediatamente Gephi detecta cuantos nodos y aristas posee el grafo. Posteriormente se genera una visualización inicial del grafo.

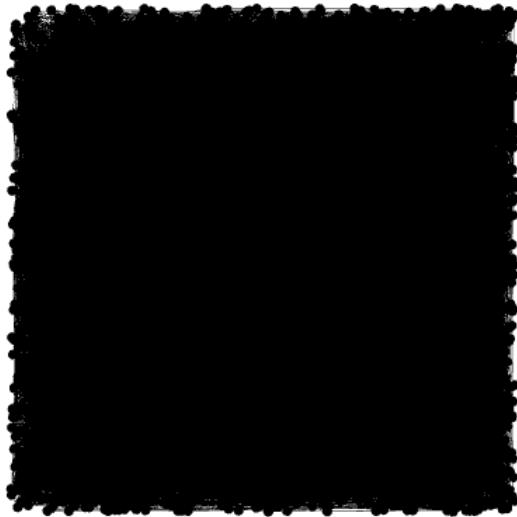


Figura 8.2: Visualización inicial del grafo.

El software produce una visión general del grafo, espacializado al azar y totalmente ilegible.

8.2. Detección de comunidades

Una red contiene subdivisiones internas denominadas comunidades. Existen métodos que permiten poner en relieve estas comunidades, que dependen de la comparación de las densidades de los enlaces dentro de un grupo, y desde el grupo hacia el resto de la red. Muchas redes complejas tienen una estructura modular subyacente, es decir, subunidades estructurales (comunidades o grupos) caracterizada por nodos altamente interconectados. La modularidad se ha introducido como una medida para evaluar la calidad de las agrupaciones.

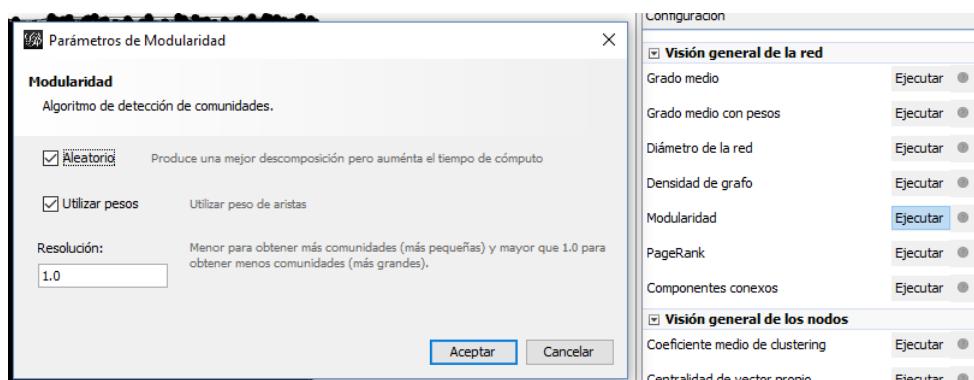


Figura 8.3: Ejecutar modularidad en Gephi.

La modularidad es una medida de la estructura de las redes o grafos. Fue diseñado para medir la fuerza de la división de una red en módulos (también llamados

grupos, agrupamientos o comunidades). Las redes con alta modularidad tienen conexiones sólidas entre los nodos dentro de los módulos, pero escasas conexiones entre nodos en diferentes módulos. La modularidad se utiliza a menudo en los métodos de optimización para la detección de la estructura comunitaria de las redes. Sin embargo, se ha demostrado que la modularidad sufre un límite de resolución y, por lo tanto, no es capaz de detectar pequeñas comunidades. Las redes biológicas, incluidos los cerebros animales, presentan un alto grado de modularidad. En este caso se detectó que existen 15 comunidades aplicando el algoritmo.

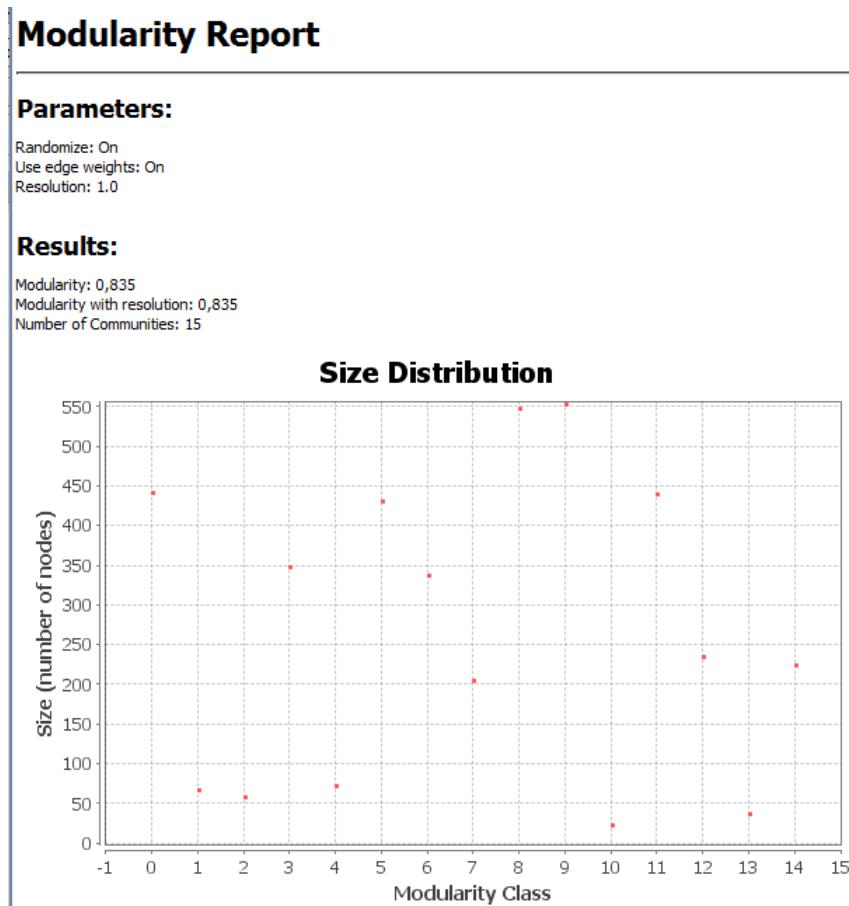


Figura 8.4: Distribución de modularidad.

Se le asigna un color a cada comunidad.

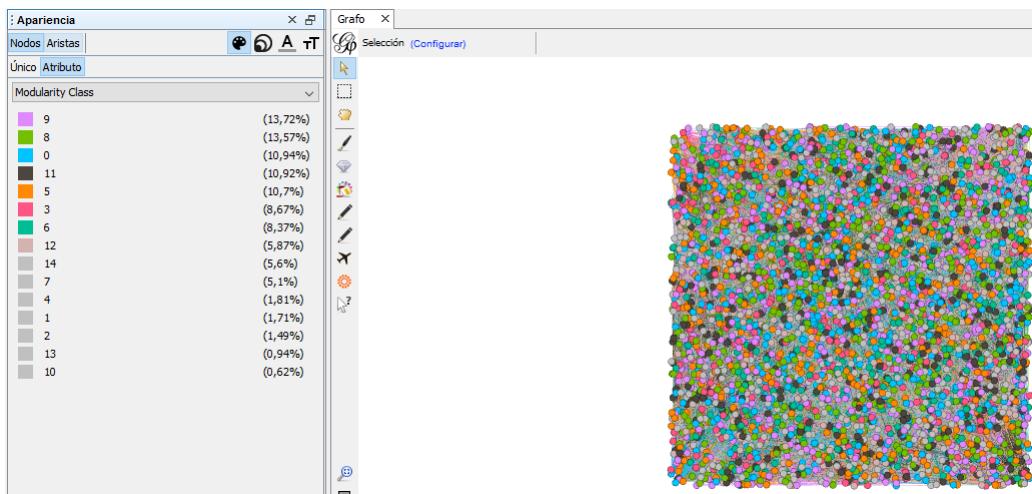


Figura 8.5: Coloración de las agrupaciones.

8.3. Selección de distribución

El propósito de las propiedades de la distribución es tomar el control de los algoritmos para realizar una representación estética más agradable. En este caso de utilizó el algoritmo de diseño, Fuerza Atlas 2 para dispersar a grupos y dar espacio en torno a nodos más grandes. Se selecciona Force Atlas 2 y se pulsa el botón ejecutar. Es en este proceso donde la herramienta separa todas las comunidades de manera visible. Se puede comprobar también como esas comunidades se separan en conjuntos de colores anteriormente configurados.

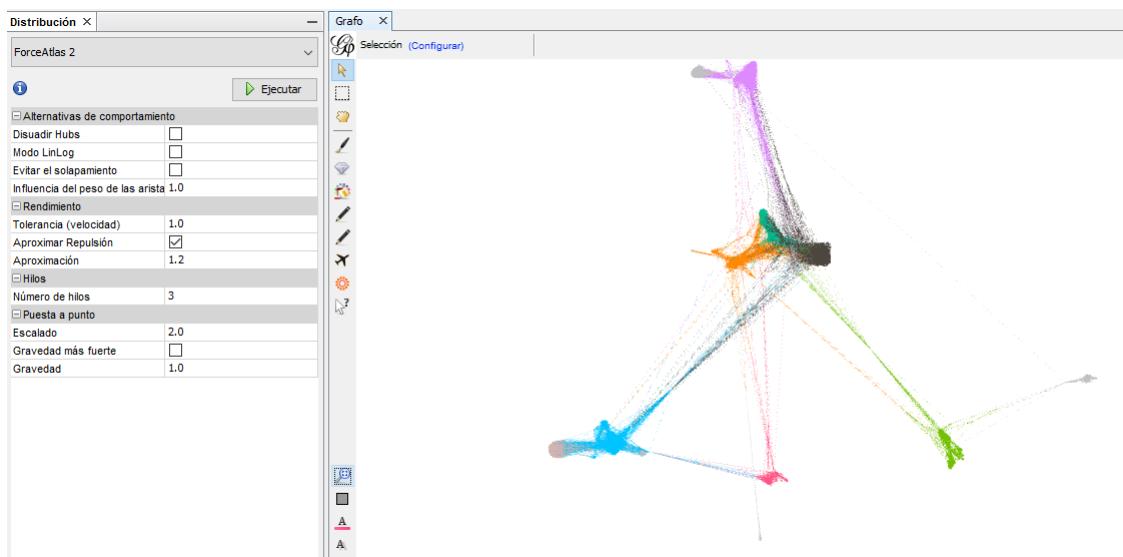


Figura 8.6: Distribución ForceAtlas 2.

Se procede a cambiar los colores de las comunidades para obtener una mejor visualización.

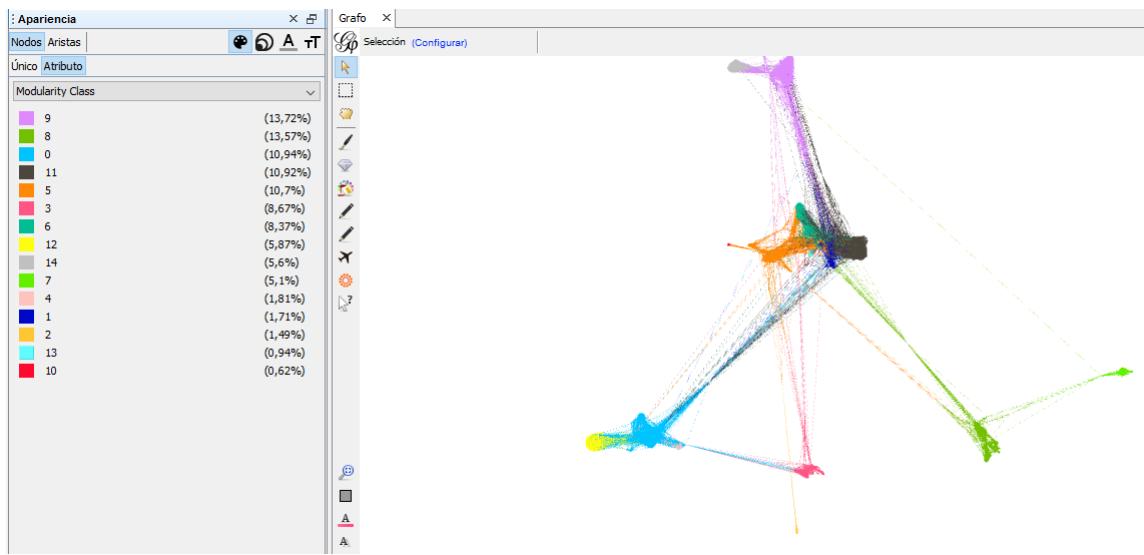


Figura 8.7: Cambiar colores de las comunidades.

Se puede visualizar como están las comunidades interconectadas y el tamaño de los conglomerados.

La distribución Fruchterman Reingold dispone a los nodos de una manera gravitacional (atracción-repulsión, de hecho, como imanes), para poder distinguir comunidades (partes más densamente conectadas de la red). Ejecutamos el algoritmo y obtenemos la siguiente gráfica.

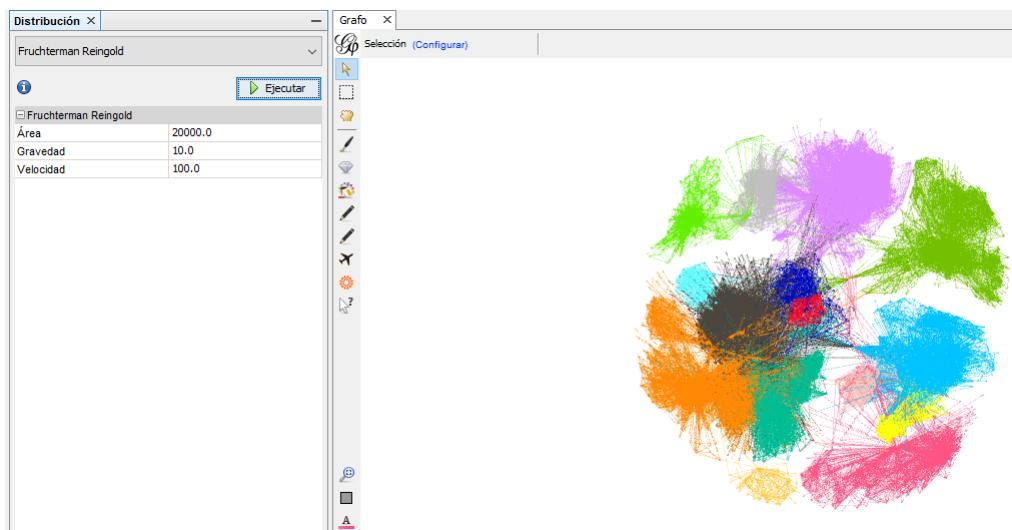


Figura 8.8: Distribución Fruchterman Reingold.

8.4. Cálculo de grados

En el siguiente gráfico se puede observar la distribución de grados del grafo, donde el eje Y representa la cantidad de nodos, y el eje X los grados.

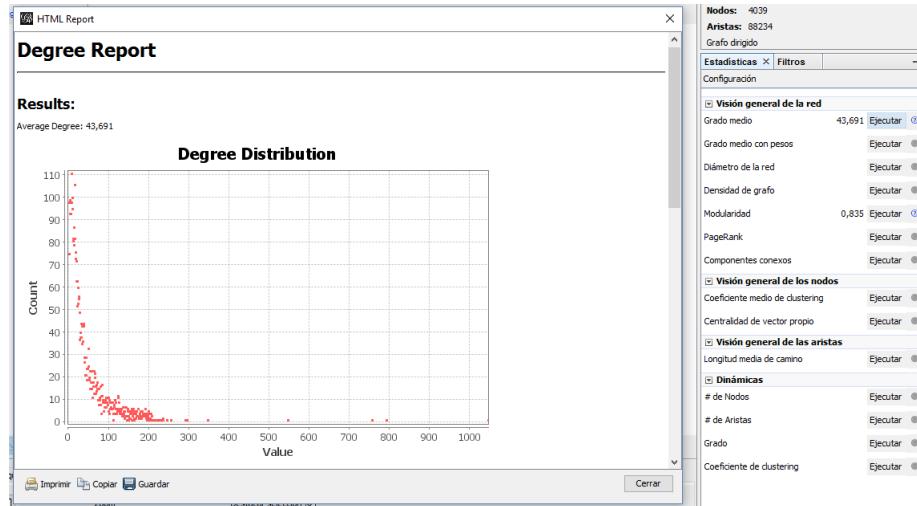


Figura 8.9: Distribución de los grados.

Distribución de los grados de entrada.

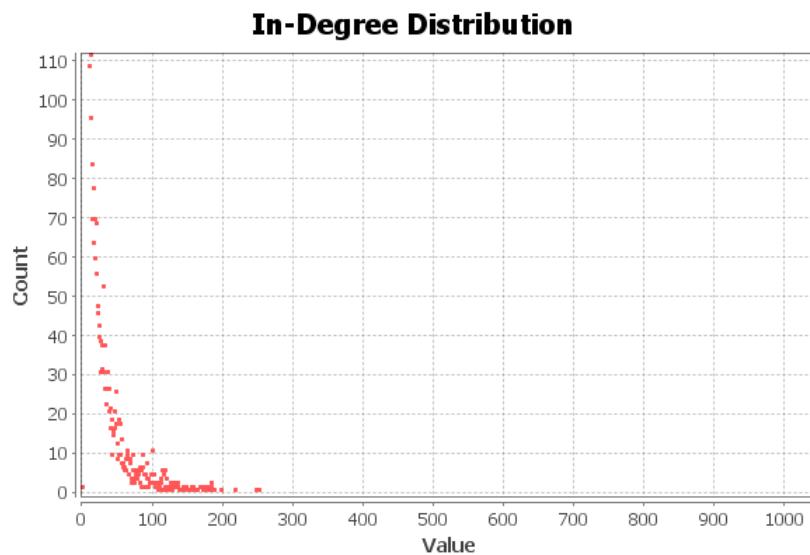


Figura 8.10: Distribución de los grados de entrada.

Distribución de los grados de salida.

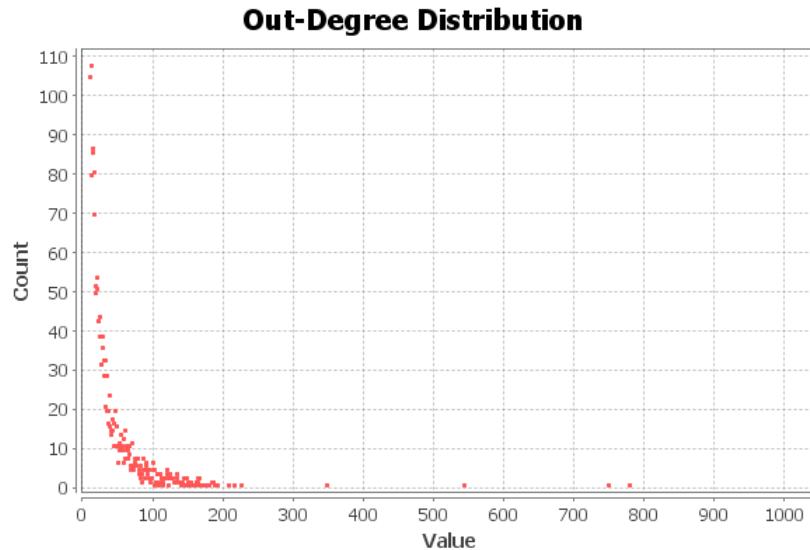


Figura 8.11: Distribución de los grados de salida.

8.5. PageRank

Gephi provee también la opción de medir la influencia de cada nodo mediante el algoritmo de PageRank.

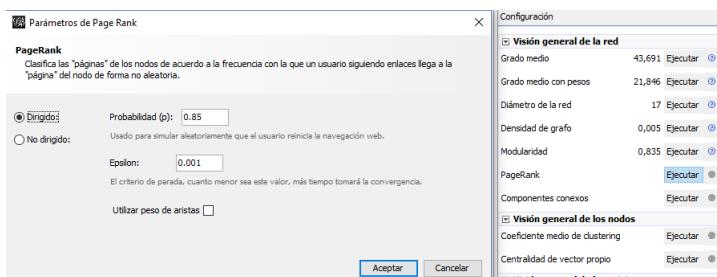


Figura 8.12: Ejecutar algoritmo de PageRank en Gephi.

Se obtiene una visualización donde mayor sea el tamaño del nodo, mayor influencia tiene sobre el grafo.

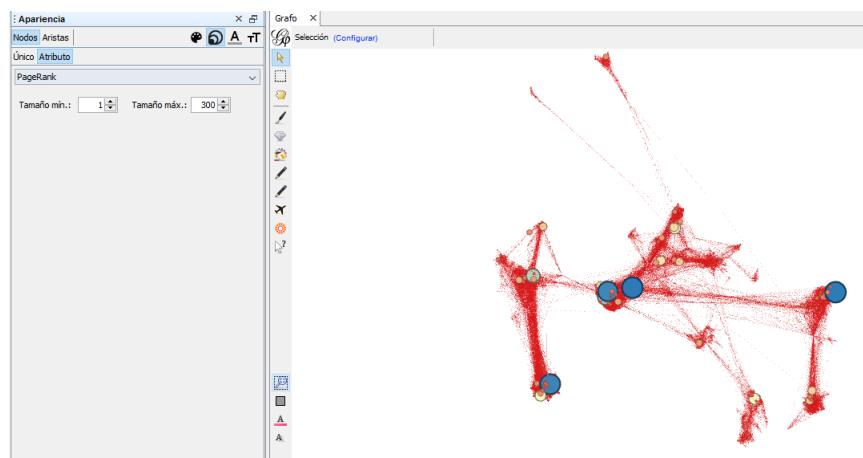


Figura 8.13: PageRank en Gephi.

Distribución del PageRank.

Parameters:

Epsilon = 0.001
Probability = 0.85

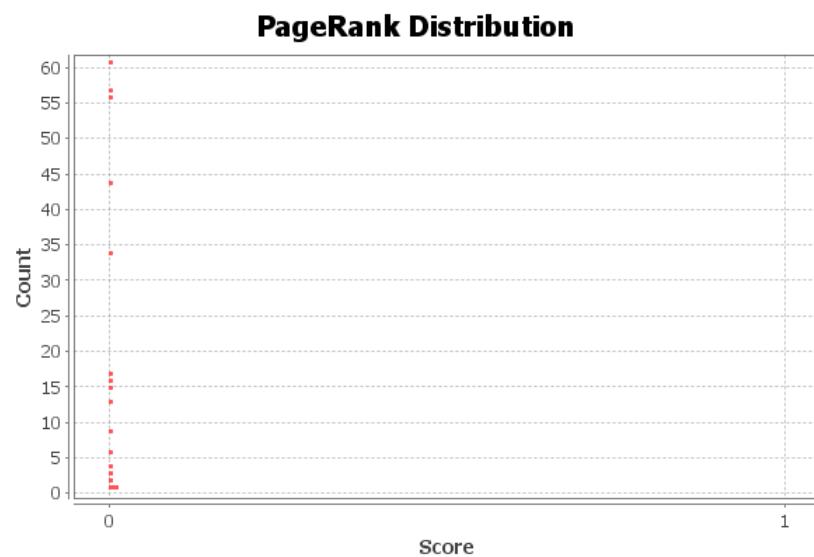
Results:

Figura 8.14: Distribución del PageRank..

CAPÍTULO 9

Conclusiones

Se logró desarrollar todo lo planteado en el alcance del trabajo y como extra se instaló un clúster con multi nodos físicos utilizando la distribución Hadoop, Cloudera, y se utilizó la herramienta para la visualización de redes Gephi.

Cloudera es una de las distribuciones de Hadoop que existen, provee un conjunto de herramientas y una serie de configuraciones que permiten su utilización. Cloudera Manager permite la instalación, utilización y monitoreo de un clúster Hadoop. Hadoop Distributed File System (HDFS) es un sistema de archivos distribuido, escalable y portátil. Spark es una herramienta cuya finalidad es procesar grandes volúmenes de datos en memoria. GraphX es un API de Spark que provee diversas implementaciones y algoritmos que pueden ser utilizados sobre grafos. Gephi es una herramienta para la visualización de redes.

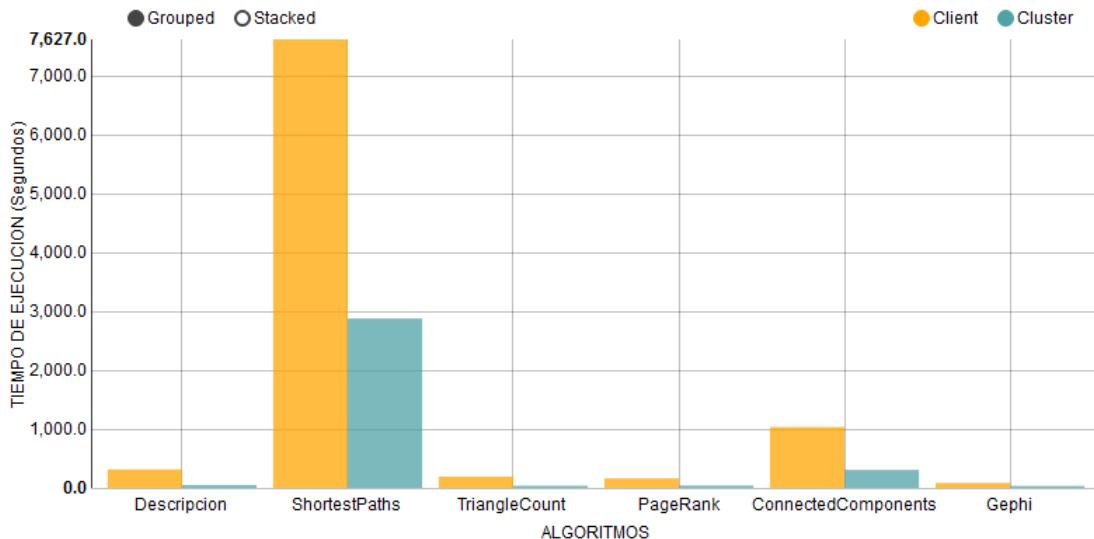


Figura 9.1: Comparación entre los tiempos de ejecución (YARN cluster vs YARN client).

Para entender el funcionamiento de cada proceso de las herramientas mencionadas anteriormente se utilizó el SandBox de Cloudera en una máquina virtual. Las implementaciones utilizando el lenguaje de programación Scala y GraphX fueron desarrolladas en el Sandbox Cloudera, y fueron probadas tanto en esta, como en el clúster multi nodo. Como se puede observar en la figura 9.1, todos los procesos de ejecución del clúster multi nodo (YARN cluster) fueron más veloces que los del SandBox Cloudera (YARN client).

Se puede concluir, que sí es posible solventar los problemas de procesamiento de Hadoop MapReduce respecto a los algoritmos iterativos, utilizando herramientas basadas en memoria, como Spark. Sí se puede almacenar estructuras de datos que representen redes en sistemas distribuidos como HDFS, es posible realizar cálculos e implementaciones sobre grafos de gran escala mediante API's como GraphX, y ejecutarlos en una infraestructura de procesamiento distribuido y paralelo. Finalmente, es factible visualizar grafos de gran escala mediante herramientas para la visualización de redes, como Gephi.

9.1. Contribución

Este trabajo de investigación contribuye principalmente con la Escuela de Computación de la Facultad de Ciencias de la Universidad Central de Venezuela, ya que posee varios factores innovadores dentro de esta institución y a nivel Nacional, ya que es una solución orientada a la Ciencia de Datos donde se utilizó la distribución Hadoop Cloudera, Apache Spark y GraphX. Se implementó un clúster funcional dentro de la Escuela de Computación, Centro de Computación Gráficas y puede ser utilizado por los docentes y alumnos de la institución para la implementación de proyectos y otras finalidades académicas. Este documento es útil como tutorial para instalar un clúster con múltiples nodos físicos, ya que la mayoría de los tutoriales disponibles en internet son de un solo nodo o nodos virtuales.

Finalmente con el proyecto desarrollado se deja la posibilidad de implementar diversas soluciones, como analizar otros tipos de redes de gran escala, utilizar otras herramientas del ecosistema Hadoop u otros, y ejecutar proyectos de todo tipo de forma distribuida y paralela.

9.2. Recomendaciones

Para ejecutar este proyecto es recomendable instalar alguna distribución Hadoop, o en su defecto instalar los componentes individualmente (HDFS, YARN, Spark). Poseer un clúster con multi nodos físicos para tener un buen rendimiento, con un clúster de un solo nodo o con multi nodos virtuales es posible realizar el procesamiento, sin embargo la velocidad de ejecución es alterada negativamente. Tener conocimientos del área de redes, lenguaje Scala, ecosistema Hadoop y Linux

para poder solventar cualquier problema que pueda ocurrir al momento de realizar configuraciones, implementaciones o instalaciones adicionales.

9.3. Trabajos Futuros

Sobre el trabajo realizado existen distintas modificaciones que pueden ser hechas para mejorar su utilidad. En primer lugar instalar el clúster en máquinas que posean características superiores a las usadas en este proyecto y añadirles más nodos. Utilizar otras herramientas del ecosistema Hadoop que puedan facilitar aún más el trabajo de los usuarios y permita expandir la variedad de problemas que puedan ser resueltos, por ejemplo, Spark SQL, Spark Streaming, MLlib, entre otros.

Generar conexiones entre Apache Spark y bases de datos orientada a grafos, y así obtener 2 grandes ventajas, una base de datos de transacciones y velocidad de procesamiento. Desarrollar implementaciones sobre diversos sistemas de procesamiento de grafo y realizar comparaciones de rendimiento, velocidad y otros. Desarrollar técnicas, herramientas o algoritmos que permitan obtener visualizaciones óptimas de grafos de gran escala.

Anexos

9.4. Implementación realizada en Scala junto a GraphX

```
package com.cloudera.sparksocialmedia

//Importar las bibliotecas
import org.apache.spark.{SparkConf, SparkContext}
// El underscore es un wildcard que indica que todo lo de esa biblioteca debe ser importado
import org.apache.spark.graphx._
import org.apache.spark.rdd._

object SparkSocialMedia extends App {

    //Generar un SparkContext
    val sc = new SparkContext(new SparkConf().setAppName("Social Media"))
    //Cargar grafo de HDFS
    // GraphLoader es un objeto de la biblioteca GraphX que contiene un metodo llamado
    // edgeListFile que permite cargar grafos de un formato .txt que contenga una lista de aristas
    // edgeListFile recibe 2 parametros, el primero es SparkContext (sc) que es instanciada
    // por la consola de Spark org.apache.spark.SparkContext que es el punto de entrada
    // para distintas funcionalidades de Spark, y el segundo parametro es el archivo .txt

    val grafo = GraphLoader.edgeListFile(sc, args(0))

    //-----ALGORITMO SHORTESTPATHS-
    println("-----ShortestPaths----- \n")
    //Obtengo todos los nodos y los almaceno en una secuencia
    val x = grafo.vertices.collect()
```

Figura 9.2: Implementación realizada en Scala junto a GraphX. Parte I.

```

val nodos = x.map(_._1).toSeq

//Ejecuto ShortestPath
//nodos.take(n) calcula la distancia de algunos nodos
//nodos calcula la distancia de todos los nodos
val shortest = lib.ShortestPaths.run(graf0,nodos.take(2)).vertices.collect()
sc.parallelize(shortest,1).saveAsTextFile(args(3))

//Calculo la distancia promedio
//Almaceno todas las distancias
val otro = shortest.map(_._2).toList

//Calculo la suma de todas las distancias
val total = for (e <- otro) yield e.values.sum

//Calculo la cantidad de distancias que existen
val tamano = for (e <- otro) yield e.size
//Calculo el promedio
val promedio = total.sum / tamano.sum

-----ALGORITMO TRIANGLECOUNT-----
println("-----TriangleCount----- \n")

//Calculo la cantidad de triangulos que posee cada nodo
val triangulos = graf0.triangleCount.vertices.sortBy(_._2,ascending=False).collect()

```

Figura 9.3: Implementación realizada en Scala junto a GraphX. Parte II.

```

sc.parallelize(triangulos,1).saveAsTextFile(args(4))

-----ALGORITMO PAGERANK-----
println("-----PageRank----- \n")

//Calculo la influencia de cada nodo y las ordeno de mayor a menor
//Cuando se ejecuta la funcion pageRank se crea otro grafo donde los vertices
//tienen como atributos los valores del PageRank.
//pageRank le asigna a cada vertice la medida de influencia que tiene contra
//el resto del grafo

//El valor 0,001 es la tolerancia que permite establecer un equilibrio entre la
//velocidad y la exactitud del resultado final.
//Si la tolerancia es muy alto el algoritmo termina rapidamente su ejecucion
//pero se obtienen resultados imprecisos.
//Si la tolerancia es muy baja el algoritmo tarda mucho tiempo en ejecutarse
//y no anade precision a los resultados.

val influyentes = grafo.pageRank(0.001).vertices.sortBy(_.value,ascending=false).collect()
sc.parallelize(influyentes,1).saveAsTextFile(args(5))

```

```

-----ALGORITMO CONNECTEDCOMPONENTS-----
println("-----ConnectedComponents----- \n")

```

Figura 9.4: Implementación realizada en Scala junto a GraphX. Parte III.

```

def extraer(s: String) = {
  val Pattern = """^.*?(\d+).egonet""".r
  val Pattern(num) = s
  num
}
//Busca y genera una lista de circulos de amigos
// Procesa cada linea y retorna un arreglo de aristas en tuplas

def get_edges_from_line(line: String): Array[(Long, Long)] = {
  val ary = line.split(":")
  val srcId = ary(0).toInt
  val dstIds = ary(1).split(" ")
  val edges = for {
    dstId <- dstIds
    if (dstId != "")
  } yield {
    (srcId.toLong, dstId.toLong)
  }
  // A subtle point: if the user is not connected to
  // anyone else then we generate a "self-connection"
  // so that the vertex will be included in the graph
  // created by Graph.fromEdgeTuples.
  if (edges.size > 0) edges else Array((srcId, srcId))
}
//-----

```

Figura 9.5: Implementación realizada en Scala junto a GraphX. Parte IV.

```

def crear_aristas(contents: String) = {
  val lines = contents.split("\n")
  val unflat = for {
    line <- lines
  } yield {
    get_edges_from_line(line)
  }
  //Necesito un arreglo de tuplas para poderselo pasar a Graph.fromEdgeTuples
  //pero tengo un arreglo de arreglos de tuplas.
  //Utilizo la funcion flatten para solventar el problema

  val flat = unflat.flatten
  flat
}
//-----
//Construyo un grafo utilizando las tuplas de aristas y ejecuto connectedComponents

def obtener_circulos(flat: Array[(Long, Long)]) = {
  val edges = sc.parallelize(flat)
  val g = Graph.fromEdgeTuples(edges,1)
  val cc = g.connectedComponents()
  cc.vertices.map(x => (x._2, Array(x._1))).
  reduceByKey( (a,b) => a ++ b).
  values.map(_.mkString(" ")).collect.mkString(" | ")
}

```

Figura 9.6: Implementación realizada en Scala junto a GraphX. Parte V.

```

val egonets = sc.wholeTextFiles(args(1))
//wholeTextFiles retorna un PairRDD con un elemento de cada archivo donde la clave es el
//ruta de la carpeta en el archivo, y el valor es el contenido del archivo

val egonet_numbers = egonets.map(x => extraer(x._1)).collect
//extract: utiliza una expresión regular para extraer el ID de usuario del nombre de archivo

val egonet_edges = egonets.map(x => crear_aristas(x._2)).collect
//crear_aristas: crea aristas entre cada amigo

val egonet_circles = egonet_edges.toList.map(x => obtener_circulos(x))

val result = egonet_numbers.zip(egonet_circles).map(x => "Egonet: " + x._1 + ".\n" + "Componentes: " + x._2 + ".\n")

sc.parallelize(List(result.mkString("\n"))).saveAsTextFile(args(6))

```

Figura 9.7: Implementación realizada en Scala junto a GraphX. Parte VI.

```

-----ALGORITMO GEPHI-----
println("-----Gephi----- \n")
//Creo un .gexf del grafo
def toGexf[VD,ED](g:Graph[VD,ED]) = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n"
+ "<gexf xmlns=\"http://www.gexf.net/1.2draft\" version=\"1.2\">\n"
+ " <graph mode=\"static\" defaultedgetype=\"directed\">\n"
| + "   <nodes>\n" + g.vertices.map(v => "      <node id=\"" + v._1
| + "\" label=\"" + v._2 + "\" />\n").collect.mkString
| + "   </nodes>\n" + "   <edges>\n" + g.edges.map(e => "      <edge source=\"" + e.srcId + "\" target=\""
| + e.dstId + "\" label=\"" + e.attr + "\" />\n").collect.mkString +
"   </edges>\n" + " </graph>\n" + "</gexf>\n"

val pw = new java.io.PrintWriter("gephi/grafos.gexf")
pw.write(toGexf(grado))
pw.close

-----ALGORITMO DESCRIPCION-----
println("-----Descripcion----- \n")
//Calculo la cantidad de vertices y aristas del grafo
val aristas = grado.numEdges
val vertices = grado.numVertices

//Calculo el maximo grado
def max(a: (VertexId, Int), b: (VertexId, Int)): (VertexId, Int) = {
if (a._2 > b._2) a else b
}

//Calculo el minimo grado
def min(a: (VertexId, Int), b: (VertexId, Int)): (VertexId, Int) = {
if (a._2 <= b._2) a else b
}

//Obtengo el maximo outdegree
val outMax = grado.outDegrees.reduce(max)

//Obtengo el maximo indegree
val inMax = grado.inDegrees.reduce(max)

```

Figura 9.8: Implementación realizada en Scala junto a GraphX. Parte VII.

```

//Obtengo el maximo indegree
val inMax = grado.inDegrees.reduce(max)

//Obtengo el minimo outdegree
val outMin = grado.outDegrees.reduce(min)

//Obtengo el minimo indegree
val inMin = grado.inDegrees.reduce(min)

//Calculo el total de triangulos, los 10 usuarios mas influyentes y la distancia promedio.
val descripcion = "Número de vértices: " + vertices + ".\n" + "Número de aristas: " +
aristas + ".\n" + "Máximo outDegrees: " + "Nodo -> " + outMax._1 + ", Grados -> " +
outMax._2 + ".\n" + "Máximo inDegrees: " + "Nodo -> " + inMax._1 + ", Grados -> " +
inMax._2 + ".\n" + "Mínimo outDegrees: " + "Nodo -> " + outMin._1 + ", Grados -> " +
outMin._2 + ".\n" + "Mínimo inDegrees: " + "Nodo -> " + inMin._1 + ", Grados -> " +
inMin._2 + ".\n" + "Total de triángulos: " + triangulos.map(_.2).toSeq.foldLeft(0)(+__) + ".\n" +
"Usuarios más influyentes: " + influyentes.take(10).toList + ".\n" +
"Distancia promedio entre todos los nodos del grafo: " + promedio + ".\n"

sc.parallelize(List((descripcion)),1).saveAsTextFile(args(2))

}

```

Figura 9.9: Implementación realizada en Scala junto a GraphX. Parte VIII.

9.5. Encontrar inconvenientes del proceso de instalación de Cloudera

Lo primero es saber el estado de nuestra instalación.

```
$> sudo ps -aef | grep yum
```

Seguir los logs de la instalación.

```
$> cd /var/log/cloudera-manager-installer/
$> tail -f install-cloudera-manager-server.log
```

Ver el estado del Cloudera Manager.

```
$> service cloudera-scm-server status
```

Ver si los procesos del servidor de la BBDD Postgres, el servidor de Cloudera, y el job java asociado se encuentran corriendo.

```
$> ps -aef | grep scm
```

9.6. Desinstalar la distribución Cloudera

Para desinstalar la distribución Cloudera del sistema operativo se puede ejecutar el siguiente comando:

```
$> sudo /usr/share/cmf/uninstall-cloudera-manager.sh
```

Sin embargo, es posible que no se desnstale adecuadamente, por lo tanto es conveniente seguir los siguientes pasos.

Detener los servicios (en caso de no detenerse, utilizar el comando kill).

```
$> sudo service cloudera-scm-server stop
$> sudo service cloudera-scm-server-db stop
```

Eliminar todo rastro de la instalación en el repositorio yum.

```
$> cd /etc/yum.repos.d
$> sudo yum remove 'cloudera-manager-*'
$> sudo rm -Rf /usr/share/cmf /var/lib/cloudera* \
/var/cache/yum/cloudera*
$> cd /etc/yum.repos.d
$> sudo yum remove 'cloudera-manager-*'
$> sudo rm -Rf /usr/share/cmf /var/lib/cloudera* \
/var/cache/yum/cloudera*
```

Limpiar la cache de yum.

```
$> sudo yum clean all
```

Eliminar los siguientes ficheros.

```
$> sudo rm /var/run/cloudera-scm-server.pid  
$> sudo rm /tmp/.scm_prepare_node.lock
```

Bibliografía

- [1] Definición de datos. <http://definicion.de/datos/>, 2016. Recuperado el 3 de Mayo de 2016.
- [2] Definición de Información. <http://www.definicionabc.com/tecnologia/informacion.php>, 2016. Recuperado el 3 de Mayo de 2016.
- [3] Sobreconceptos. <http://sobreconceptos.com/conocimiento>, 2016. Recuperado el 3 de Mayo de 2016.
- [4] Jure Leskovec Anand Rajaraman and Jeffrey D. Ullman. *Mining of Massive Datasets*. Palo Alto, CA, 2014.
- [5] Machine Learning | Stanford Online. <http://online.stanford.edu/course/machine-learning>, 2016. Recuperado el 3 de Mayo de 2016.
- [6] Intro to Artificial Intelligence Course and Training Online. <https://www.udacity.com/course/intro-to-artificial-intelligence--cs271>, 2016. Recuperado el 5 de Mayo de 2016.
- [7] What is business intelligence (BI)? <http://searchdatamanagement.techtarget.com/definition/business-intelligence>, 2016. Recuperado el 5 de Mayo de 2016.
- [8] What is Data Science? <https://datascience.berkeley.edu/about/what-is-data-science/>, 2016. Recuperado el 1 de Mayo de 2016.
- [9] What is big data? <http://searchcloudcomputing.techtarget.com/definition/big-data-Big-Data>, 2016. Recuperado el 1 de Mayo de 2016.
- [10] Obs business school. <http://www.obs-edu.com/noticias/estudio-obs>, 2016. Recuperado el 1 de Mayo de 2016.
- [11] What is database? <http://searchsqlserver.techtarget.com/definition/database>, 2016. Recuperado el 5 de Mayo de 2016.
- [12] ¿Qué es Base de datos relacional? <http://searchdatacenter.techtarget.com/es/definicion/Base-de-datos-relacional>, 2016. Recuperado el 5 de Mayo de 2016.

- [13] NOSQL Databases. <http://nosql-database.org/>, 2016. Recuperado el 5 de Mayo de 2016.
- [14] Datawarehouse. http://www.sinnexus.com/business_intelligence/datawarehouse.aspx, 2016. Recuperado el 5 de Mayo de 2016.
- [15] Lenguajes de programación. <http://www.lenguajes-de-programacion.com/lenguajes-de-programacion.shtml>, 2016. Recuperado el 7 de Julio de 2016.
- [16] R: The R Project for Statistical Computing. <https://www.r-project.org/>, 2016. Recuperado el 7 de Julio de 2016.
- [17] RStudio. <https://www.rstudio.com/>, 2016. Recuperado el 7 de Julio de 2016.
- [18] Gabor Csardi. *Network Analysis and Visualization*. R, 2015.
- [19] What is Java? <http://searchsoa.techtarget.com/definition/Java>, 2016. Recuperado el 7 de Julio de 2016.
- [20] The Scala Programming Language. <http://www.scala-lang.org/>, 2016. Recuperado el 21 de Julio de 2016.
- [21] What is Python? <http://searchenterpriselinux.techtarget.com/definition/Python>, 2016. Recuperado el 7 de Julio de 2016.
- [22] Welcome to Apache Hadoop. <http://hadoop.apache.org/>, 2016. Recuperado el 21 de Julio de 2016.
- [23] Apache Commons. <https://commons.apache.org/>, 2016. Recuperado el 21 de Julio de 2016.
- [24] HDFS Architecture Guide. https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html, 2016. Recuperado el 21 de Julio de 2016.
- [25] Apache Hadoop 2.7.2 – Apache Hadoop YARN. <http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html>, 2016. Recuperado el 22 de Julio de 2016.
- [26] MapReduce Tutorial. https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html, 2016. Recuperado el 22 de Julio de 2016.
- [27] Apache Accumulo. <https://accumulo.apache.org/>, 2016. Recuperado el 22 de Julio de 2016.
- [28] Apache HBase. <https://hbase.apache.org/>, 2016. Recuperado el 22 de Julio de 2016.
- [29] Apache Hive TM. <https://hive.apache.org/>, 2016. Recuperado el 22 de Julio de 2016.

- [30] Apache Storm. <http://storm.apache.org/>, 2016. Recuperado el 22 de Julio de 2016.
- [31] Apache Mahout: Scalable machine learning and data mining. <http://mahout.apache.org/>, 2016. Recuperado el 22 de Julio de 2016.
- [32] Falcon - Feed management and data processing platform. <https://falcon.apache.org/>, 2016. Recuperado el 23 de Julio de 2016.
- [33] Welcome to Apache Flume . <https://flume.apache.org/>, 2016. Recuperado el 23 de Julio de 2016.
- [34] Sqoop. <http://sqoop.apache.org/>, 2016. Recuperado el 23 de Julio de 2016.
- [35] Ambari. <https://ambari.apache.org/>, 2016. Recuperado el 23 de Julio de 2016.
- [36] Apache ZooKeeper. <http://zookeeper.apache.org>, 2016. Recuperado el 23 de Julio de 2016.
- [37] Apache Oozie Workflow Scheduler for Hadoop. <http://oozie.apache.org/>, 2016. Recuperado el 23 de Julio de 2016.
- [38] Knox Gateway – REST API Gateway for the Apache Hadoop Ecosystem. <https://knox.apache.org/>, 2016. Recuperado el 23 de Julio de 2016.
- [39] Apache Ranger - Introduction. <http://ranger.apache.org/>, 2016. Recuperado el 23 de Julio de 2016.
- [40] Mike Frampton. *Mastering Apache Spark*. Packt, 2016.
- [41] DAG vs MapReduce - Mammoth Data. <http://mammothdata.com/dag-vs-mapreduce/>, 2015. Recuperado el 2 de Junio de 2016.
- [42] org.apache.spark.rdd.RDD. <https://spark.apache.org/docs/0.8.1/api/core/org/apache/spark/rdd/RDD.html>, 2016. Recuperado el 2 de Junio de 2016.
- [43] The modern platform for data management and analytics - Cloudera. <https://www.cloudera.com/>, 2016. Recuperado el 8 de Junio de 2016.
- [44] Hortonworks: Open and Connected Data Platforms. <http://hortonworks.com/>, 2016. Recuperado el 8 de Junio de 2016.
- [45] MapR: Converged Data Platform. <https://www.mapr.com/>, 2016. Recuperado el 8 de Junio de 2016.
- [46] Grafo. <http://gaussianos.com>, 2016. Recuperado el 7 de Julio de 2016.
- [47] Neo4j: The World's Leading Graph Database. <https://neo4j.com/>, 2016. Recuperado el 13 de Junio de 2016.

- [48] Sparsity-technologies (DEX/Sparksee). <http://www.sparsity-technologies.com/>, 2016. Recuperado el 13 de Junio de 2016.
- [49] Oracle Spatial and Graph. <http://www.oracle.com/technetwork/database/options/spatialandgraph/overview/spatialandgraph-1707409.html>, 2016. Recuperado el 13 de Junio de 2016.
- [50] SNAP: Ringo. <http://snap.stanford.edu/ringo>. Recuperado el 20 de Mayo de 2016.
- [51] PEGASUS: Peta-Scale Graph Mining System. <http://www.cs.cmu.edu/~pegasus/>, 2016. Recuperado el 20 de Mayo de 2016.
- [52] Mazerunner. <https://neo4j.com/blog/using-apache-spark-neo4j-big-data-graph-analytics/>, 2016. Recuperado el 20 de Mayo de 2016.
- [53] Fast, Scalable Machine Learning Platform. <https://turi.com/index.html>, 2016. Recuperado el 20 de Mayo de 2016.
- [54] Giraph - Welcome To Apache Giraph! <http://giraph.apache.org/>, 2016. Recuperado el 20 de Mayo de 2016.
- [55] Gephi - The Open Graph Viz Platform. <https://gephi.org/>, 2016. Recuperado el 2 de Julio de 2016.
- [56] Graphviz - Graph Visualization Software. <http://www.graphviz.org/>, 2016. Recuperado el 2 de Julio de 2016.
- [57] D3.js - Data-Driven Documents. <https://d3js.org/>, 2016. Recuperado el 2 de Julio de 2016.
- [58] "GraphStream - A Dynamic Graph Library. <http://graphstream-project.org/>, 2016. Recuperado el 13 de Junio de 2016.
- [59] Michael S. Malak y Frank Robin East. *Spark GraphX in action*. Manning, Shelter Island, New York, 2016.