Universidad Zaragoza

Trabajo Fin de Grado

UTILIZACIÓN Y ANÁLISIS DE HERRAMIENTAS DE BIG DATA PARA EL ESTUDIO DE REGISTROS DE CONEXIONES A INTERNET

USAGE AND ANALYSIS OF BIG DATA TOOLS FOR THE STUDY OF INTERNET CONNECTION LOGS

Autor

Manuel Hernández Rodrigo

Director

Idilio Drago

Escuela de ingeniería y arquitectura

2016

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

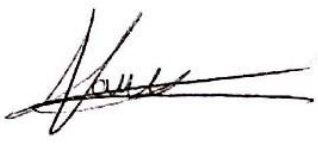
D./Dª. MANUEL HERNÁNDEZ RODRIGO,

con nº de DNI 72892454-A en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) GRADO, (Título del Trabajo) Utilización y análisis de herramientas de Big Data para el estudio de registros de conexiones a internet,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 22 de Noviembre de 2016



Fdo: MANUEL HERNÁNDEZ RODRIGO

UTILIZACIÓN Y ANÁLISIS DE HERRAMIENTAS DE BIG DATA PARA EL ESTUDIO DE REGISTROS DE CONEXIONES A INTERNET

RESUMEN

El objetivo de la tecnología *Big Data* es el estudio de grandes cantidades de datos cuyo análisis tomaría demasiado tiempo en una base de datos tradicional. En este trabajo se emplean estas tecnologías para realizar el estudio de una red de tráfico Internet. Para ello se utilizan las herramientas de *Big Data* pertenecientes a Apache Hadoop: MapReduce, Spark y Hive.

Estas herramientas se encuentran funcionando sobre un *cluster* de ordenadores ubicado en la Universidad Politécnica de Turín. En esta red se genera una monitorización del tráfico perteneciente al protocolo internet (IP) mediante un *sniffer* que crea los registros de tráfico sobre los que se trabaja.

El primer problema que se plantea es realizar un estudio de las características de las herramientas pertenecientes a Apache Hadoop en su uso para el análisis de los registros de tráfico de red almacenados. Para ello se realizan una serie de pruebas que permiten comprobar sus resultados frente a diferentes tipos de análisis.

Al finalizar el estudio de estas herramientas, se realiza un análisis sobre el tráfico IP almacenado para caracterizar los protocolos utilizados en la red y el tráfico generado. Debido a que la mayoría del tráfico registrado pertenece al protocolo de transferencia de hipertexto (HTTP), se estudia la relación que tiene en los servicios web modernos el dominio visitado y las direcciones IP utilizadas.

Índice general

Índice de figuras y tablas

CAPÍTULO 1

Introducción

En esta sección se ofrece una visión general del proyecto de fin de grado, presentando los objetivos y un resumen del trabajo realizado.

1.1 Visión general

Actualmente nos encontramos en un entorno en el cual se está creando y almacenando información constantemente y cada vez en mayor cantidad. Esta información se genera de diversas maneras, como puede ser:

-Por usuarios mediante el uso de web y móvil, mediante Facebook, Twitter, YouTube, etc.,

-Por computación para fines científicos o sanitarios.

-Por servidores creando archivos de registro de datos, almacenando información de usuarios, operaciones, etc.

- Por el internet de las cosas (*Internet of Things*).

Todos estos sistemas generan cada vez un volumen de datos mayor y que sigue con una tendencia al alza, con lo que aumentará cada vez más la cantidad de datos almacenados.

No es posible conocer con seguridad la cantidad de datos almacenados electrónicamente, pero una estimación realizada por IDC [2] coloca el tamaño de estos datos a 4.4 Zettabytes en 2013 con una tendencia de crecimiento por un factor 10 para 2020 por lo que se espera que alcance los 44 Zettabytes. Un Zettabyte corresponde a 10^{21} Bytes, equivalente a mil millones de terabytes.

Esta tendencia a almacenar gran cantidad de información ha llevado al desarrollo de nuevas tecnologías para poder realizar análisis sobre estos datos, lo que provoca la aparición del *Big Data*.

Una de las definiciones que podemos obtener de *Big Data* es [3] “Datos cuya extensión, diversidad y complejidad necesitan de nuevas arquitecturas, algoritmos y técnicas de análisis para gestionarlos y extraer el valor y conocimiento de ello”. Esto implica que

para el análisis de estos datos es necesaria una nueva tecnología. Esta se basa en el uso de *clusters* para realizar el análisis de los datos en paralelo y con ello reducir el tiempo de análisis.

Esta tecnología queda comúnmente caracterizada por las llamadas tres 'V' que corresponden con velocidad, volumen y variedad, ya que se caracteriza por obtener resultados con una velocidad mayor, datos de volumen más grande y de tipo muy variado.

Este Trabajo de Fin de Grado (TFG) se centra en realizar un estudio sobre el tráfico de red que se genera en la Universidad Politécnica de Turín. Debido a la gran cantidad de datos producidos en la red, se hace uso de técnicas de *Big Data* que posibilitan su análisis. Este trabajo se enmarca dentro del grupo de trabajo BigData@PoliTO perteneciente a la universidad Politécnica de Turín. [1]

1.2 Usos en la actualidad

La tecnología utilizada en este trabajo está siendo cada vez más utilizada en la actualidad con una tendencia claramente ascendente. Estas tecnologías se empiezan a implementar en todos los sectores empresariales como recoge un informe realizado por OBS [10] en el que indica como en el año 2014 se registra un aumento de las inversiones en Big Data en todos los sectores empresariales.

Uno de las mayores utilidades de Big Data para las empresas es utilizar estas herramientas para analizar los datos que generan los clientes y con ello comprenderlos mejor. Para ello, a partir de los datos analizados se generan modelos predictivos del comportamiento del cliente. Estos modelos son utilizados por las empresas para buscar un beneficio, por ejemplo, mediante publicidad personalizada para cliente.

Otra aplicación interesante de estas técnicas es su uso en el sector de la salud. Según se recoge en una noticia de la cnbc [11], un hospital de Toronto utiliza estas técnicas para monitorizar bebés en la unidad de cuidados intensivos de recién nacidos de manera prematura. El desarrollo de una serie de algoritmos ha permitido que mediante el análisis de la respiración y el pulso de un bebe se puedan predecir infecciones antes de que aparezcan.

Este trabajo se centra sobre la utilización de estas técnicas para analizar tráfico de red. En este ámbito Big Data también tiene una importante aplicación en este sector. Por ejemplo, se utiliza para detectar y prevenir ciberataques mediante monitorizaciones del tráfico de red, utilizando patrones y algoritmos para detectar anomalías, o para realizar análisis de contenido web e información obtenida a partir de las redes sociales.

1.3 Objetivo

Este proyecto comprende el estudio de determinadas herramientas de *Big Data* pertenecientes a la plataforma Apache Hadoop: como son Java MapReduce, Apache Hive y Apache Spark y su uso para realizar un análisis de registros de tráfico de red. La primera parte se centra en las diferencias obtenidas por el uso de cada una de estas herramientas. Para ello, se hace uso de un *cluster,* o conjunto de ordenadores, instalado en la universidad Politécnica de Turín que permite trabajar con grandes volúmenes de datos.

En la red de la universidad se ha instalado un analizador del tráfico que circula sobre ella. Este *sniffer* analiza las cabeceras de los paquetes que atraviesan la red para realizar registros de tráfico con la información obtenida de ellos. La herramienta utilizada como *sniffer* es Tstat, un programa desarrollado por la misma universidad.

Mediante las herramientas de *Big Data* nombradas se realiza un estudio de las características del tráfico generado en la red y de diversos servicios que pueden observarse en los registros de red.

El disponer de una alta cantidad de usuarios pertenecientes a la red, sobre quince mil, genera un gran volumen de tráfico para analizar y extraer conclusiones sobre el tipo de tráfico que caracteriza las conexiones a Internet.

En la última parte del trabajo se realiza un estudio de los servicios web modernos utilizando para ello la relación existente entre direcciones de red IP y los dominios visitados.

1.4 Organización de la memoria

La memoria consta aparte de este primer capítulo donde se introduce el trabajo realizado, consta de cinco capítulos más y dos apéndices que son descritos a continuación:

- **Capítulo 2: Herramientas utilizadas**: este capítulo describe las herramientas de Apache Hadoop utilizadas en el trabajo y la herramienta de monitorización Tstat.
- **Capítulo 3: Arquitectura del sistema**: En este capítulo se introducen las características de la red utilizada y la estructura del *cluster.*
- **Capítulo 4: Aplicación de las herramientas de Big Data para el análisis de red:** Este capítulo recoge los análisis de rendimiento realizados sobre las diferentes herramientas de *Big Data.*
- **Capítulo 5: Análisis del tráfico de la red.** En este capítulo se recogen los resultados obtenidos al realizar diferentes análisis sobre los registros de tráfico disponibles.

- **Capítulo 6: Conclusiones y líneas futuras:** En este capítulo se recogen las conclusiones finales del trabajo y describe brevemente las posibles líneas futuras a seguir.
- **Anexo 1: *Scripts*:** Este anexo recoge los *scripts* de Linux utilizados.
- **Anexo 2: Protocolos IP, TCP, HTTP Y DNS:** En este anexo se realiza una explicación de los protocolos utilizados.

CAPÍTULO 2

Herramientas utilizadas

En esta sección se realiza una introducción teórica de los elementos utilizados.

2.1 Introducción

En este trabajo se hace uso de las herramientas de *Big Data* sobre tráfico almacenado perteneciente a Internet. Para entender las herramientas utilizadas se comienza por realizar un análisis teórico de cada una de las herramientas envueltas en el proyecto y algunos de los análisis y medidas que se recogen en Internet.

2.1.1 Mediciones en Internet

Internet se conoce como la red global que permite la interconexión de máquinas distribuidas por todo el mundo basada en el protocolo TCP (*Transport Layer Protocol*) /IP (Internet Protocol). Es la red más utilizada y numerosos servicios se utilizan sobre ella, siendo el más utilizado el servicio Web.

La monitorización de estas redes para su estudio es común para caracterizar el comportamiento de las redes. La forma más común de clasificar los métodos de medida de tráfico es distinguir entre pasivos y activos. La medición activa implica la inyección de tráfico en la red para sondear ciertos dispositivos de red (por ejemplo, PING) o para medir propiedades de red tales como tiempos de ida y vuelta (RTT) (por ejemplo, traceroute), retardo unidireccional y ancho de banda máximo. La observación del tráfico de red, denominada medición pasiva o monitoreo, no es intrusiva y no cambia el tráfico existente. La monitorización de la red se realiza en una ubicación específica de esta para realizar los análisis correspondientes. Este tipo de medición es la más adecuada para el análisis de las propiedades de tráfico de Internet ya que no varía el tráfico existente.

Debido al alto volumen de tráfico generado por Internet es común realizar el estudio del tráfico agrupando los datos mediante flujos de red. Un flujo puede ser descrito como una secuencia de paquetes intercambiados entre equipos, definido por ciertos campos dentro de las cabeceras de red y de transporte. Así, en lugar de registrar cada paquete

individual, se almacenan registros de flujo, que contienen información relevante sobre cada flujo específico.

Sobre estos registros se realizan análisis variados, por ejemplo, estudiar el tráfico HTTP (*Hypertext Transfer Protocol), peer* to *peer* o DNS (*Domain Name System).* También se utilizan las medidas de Internet para detectar fallos o patrones de ataque a una red.

Desde el punto de vista de los operadores de red, las medidas que se realizan se encuentran interesadas principalmente en la latencia y la capacidad utilizada de los caminos sobre su red. Por lo tanto, pueden realizar actividades de medición que observen activamente tanto el uso de la red (tráfico en la red) como las capacidades de red (capacidad, latencia, etc.). Debido a que todo esto está dentro de su propia red, los proveedores de servicios de Internet (ISP) pueden interpretar los datos para comprender dónde se esperan o los problemas inesperados con las conexiones de red y tratarlos en consecuencia.

2.1.2 Herramientas de procesado de *Big Data*

Como se ha comentado, *Big Data* comprende las herramientas utilizadas para procesado de grandes cantidades de datos. Estos sistemas almacenan una gran cantidad de datos y se encuentran diferentes modelos para gestionar y analizar estos en una forma escalable. Comúnmente se representa por las tres 'V':

• **Volumen**: un sistema Big Data es capaz de almacenar una gran cantidad de datos mediante infraestructuras escalables y distribuidas. En los sistemas de almacenamiento actuales empiezan a aparecer problemas de rendimiento al tener cantidades de datos del orden de magnitud de petabytes. Big Data está pensado para trabajar con estos volúmenes de datos.

• **Velocidad**: una de las características más importantes es el tiempo de procesado y respuesta sobre estos grandes volúmenes de datos, obteniendo resultados en tiempo real y procesándolos en tiempos muy reducidos.

• **Variedad**: las nuevas fuentes de datos proporcionan nuevos y distintos tipos y formatos de información que un sistema Big Data es capaz de almacenar y procesar sin tener que realizar un preproceso para estructurar o indexar la información.

Se puede hablar de una clasificación de los tipos de datos según sea su naturaleza u origen que también ayuda a entender mejor el porqué de la evolución de los sistemas de explotación de la información hacia *Big Data*:

• **Datos estructurados**: es información ya procesada, filtrada y con un formato estructurado. Es el tipo de datos que más se usan hoy en día.

• **Datos semi-estructurados**: es información procesada y con un formato definido, pero no estructurado. De esta manera se puede tener la información definida, pero con una estructura variable.

• **Datos no estructurados**: es información sin procesar y que puede tener cualquier estructura. Se puede encontrar cualquier formato: texto, imagen, vídeo, código, etc. Como son directorios de *logs* de aplicaciones o información colgada en las redes sociales.

Las diferentes herramientas de *Big Data* buscan resolver sus dos principales problemas*:* cómo gestionar el almacenamiento de los datos y el modelo de programación utilizado para procesar y analizar los datos.

Uno de los primeros modelos de programación utilizado es el creado por Google: Map Reduce. Este modelo se explica en el apartado 2.2.2. El desarrollo de Map Reduce permite dividir los datos a analizar entre un *cluster* para reducir sus tiempos de análisis.

Pero este modelo tiene limitaciones ya que no permite su uso en tiempo real y no obtiene buenos resultados para algoritmos de grafos iterativos. Para solventar estas limitaciones se desarrollan nuevas herramientas como Spark, que permite ser ejecutado en *streaming* en tiempo real e introduce un nuevo método para el análisis de datos.

También aparecen nuevas herramientas que incorporan diversas funcionalidades como:

• Kafka: Utilizado para datos a tiempo real y aplicaciones de streaming.

• Hive: Utilizado para consulta y gestión de grandes conjuntos de datos que residen en almacenamiento distribuido sobre un lenguaje similar a SQL.

• Oozie: Permite gestionar un flujo de trabajo específico mediante un DAG (Gráfico acíclico dirigido).

El trabajo se ha decidido centrar en las herramientas disponibles en Apache Hadoop como son Map Reduce, Spark y Hive. Se ha elegido estas puesto que son algunas de las herramientas más utilizadas y se encuentran dentro de la plataforma de Hadoop disponible en el *cluster* utilizado.

2.2 Apache Hadoop

[3] Apache Hadoop ofrece una librería de *software (framework of software)* cuyo objetivo es crear una estructura que permita el procesado de manera distribuida de grandes conjuntos de datos a partir de *clusters* de ordenadores que utilizan modelos de programación simples. Está diseñada para poder funcionar tanto en servidores locales

como en agrupaciones de miles de máquinas. En lugar de depender de hardware para ofrecer alta disponibilidad, la librería en si misma está diseñada para detectar y controlar los errores en la capa de aplicación.

El proyecto incluye los siguientes módulos:

- Hadoop *Common*: Proporciona acceso a los sistemas de archivos soportados por Hadoop
- Hadoop *Distributed File System* (HDFS)
- Hadoop YARN
- Hadoop MapReduce

Y proyectos relacionados como SPARK y Hive en los cuales trabajaremos.

2.2.1 Hadoop *Distributed File System* (HDFS)

Hadoop *Distributed File System* es un sistema de archivos diseñado para almacenar archivos de gran tamaño funcionando sobre *clusters*. Los datos son divididos en diferentes bloques (*chunks*) y distribuidos a través de los nodos del *cluster*. Estos bloques son de gran tamaño, en nuestro caso utilizamos bloques de 64 MB. Con esto se pretende reducir el número de búsquedas para poder realizar la lectura de los datos.

Cada uno de estos bloques se encuentra replicado tres veces en diferentes nodos, dos pertenecientes al mismo *rack* y otro en diferente *rack*. Esto contribuye a disminuir los problemas de disponibilidad de los datos por fallo en algún nodo.

Tiene un único nodo central llamado *NameNode* que contiene un directorio en memoria con la información de los bloques que forman un archivo además de dónde se encuentran estos almacenados en el *cluster*. Para realizar la lectura de un fichero, se realiza una pregunta al nodo central para conocer los bloques que lo forman y su localización. El servidor central proporciona esta información al usuario que se conecta directamente a los servidores que contienen los datos (*DataNode*).

Puesto que *Hadoop* está diseñado para funcionar sobre *clusters* para los cuales la probabilidad de fallo en un nodo puede ser alta, sobre todo en *clusters* de gran tamaño, HDFS nos permite continuar con el trabajo en una situación en la que encontremos fallos, ya que automáticamente los corrige. Este proceso de corrección de errores es transparente al usuario.

En nuestro sistema tenemos implementadas varias formas para permitir el acceso al directorio HDFS. Se puede acceder mediante una conexión a internet al servidor central ubicado en https://bigdatalab.polito.it, mediante diferentes interfaces:

- Línea de comandos mediante el comando hdfs.

- Interfaz web básica creada por *Apache Hadoop*.

- Interfaz web desarrollada por Apache *Cloudera* llamada *Hue* en el puerto 8080.

Generalmente, para establecer un enlace seguro con el *cluster* utilizaremos el protocolo *Secure Shell* (SSH), que permite acceder a máquinas remotas y utilizar en ellas un intérprete de comandos. El comando *hdfs* nos permitirá realizar diversas acciones como leer, modificar, escribir o borrar contenido del sistema distribuido de archivos. Todas las funciones disponibles se pueden obtener de la página web de Hadoop [9].

2.2.2 MapReduce

MapReduce es un modelo de programación que permite trabajar en paralelo con grandes cantidades de datos en sistemas de memoria distribuida mediante *clusters,* basado en un modelo de programación maestro/esclavo.

Hadoop MapReduce ha sido diseñado para funcionar sobre el lenguaje de programación *Java*.

El *framework* de Hadoop coordina la ejecución del programa de MapReduce con la ejecución en paralelo del programa, planificando las divisiones del trabajo (*subtask*) y la sincronización del mismo.

De la gestión de los trabajos MapReduce y las tareas individuales se encarga el *JobTracker* ubicado en el nodo maestro. En los nodos esclavo se encuentra el llamado *TaskTracker* que inicia y monitoriza las tareas asignadas al nodo.

Para el funcionamiento de un programa el cliente envía el código a ejecutar al nodo maestro quien a su vez se encarga de copiar éste en los nodos en los que se ejecutará. Este código es generalmente de un tamaño pequeño, mucho menor que el tamaño de los datos y es ejecutado en cada uno de los nodos esclavos que selecciona el *JobTracker*. Todo este proceso es transparente al usuario que solamente se encarga de enviar el código al nodo maestro.

Un programa de MapReduce se compone de tres partes principales:

-La clase *Driver*: Está caracterizada por contener el método *main*, el cual acepta los argumentos desde la línea de comandos. También configura el trabajo (*job*), lo entrega al *cluster* y coordina el flujo de trabajo de la aplicación.

-La clase *Mapper*: Caracterizada por el método *map*, mapea los datos iniciales del fichero de entrada a pares *key/value.* Se ejecuta en los nodos esclavo. Esta operación se realiza en paralelo en los nodos esclavo.

-La clase *Reducer*: Caracterizada por el método *reducer*, el cual procesa los pares intermedios y emite pares *key,value*. Para generar los pares de salida, la clase reducer agrupa previamente los valores de los pares de entrada que contienen la misma *key*. Sobre este conjunto de valores se aplica la función determinada en la clase *reducer* y se genera un par de salida para cada grupo con el mismo valor de *key*. Cada grupo de pares *key,value* puede ser procesado en paralelo para reducir el tiempo de ejecución del programa.

MapReduce garantiza que la entrada a cada *reducer* se encuentra ordenada mediante la clave (*key*). Para ello realiza una función de ordenado (*Shuffle and sort*) que no requiere estar definida en el programa. Esta función se encarga de ordenar el par de salida de la fase de mapeo mediante valor de la clave, y de agrupar los valores que contengan la misma clave en la entrada a la tarea *reduce*, creando un grupo con una única clave y una lista de valores (*values*).

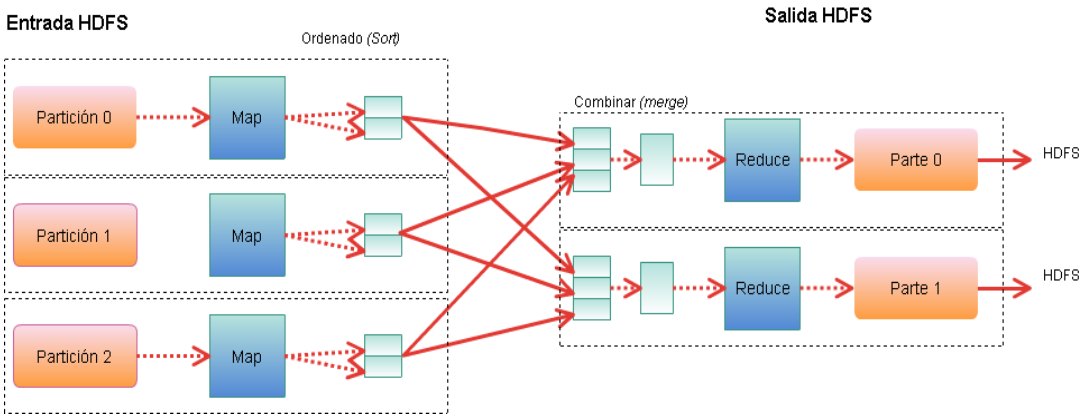


Figura 0-1: Tarea MapReduce con múltiples *reducers*.

MapReduce sólo permite realizar las operaciones de *map* y *reduce*. Todo programa se basa en estas dos operaciones para su solución.

Los resultados intermedios a cada *mapper* son agrupados bajo la misma *key* y enviados por la red a los nodos encargados de la función *reduce*. También permite la introducción en el programa de un *combiner*, mediante el cual cada nodo agrupa y reduce los datos bajo la misma *key* antes de enviar la salida del *mapper*, esto permite disminuir la cantidad de datos que se envía a la red.

En caso de producirse algún fallo en el nodo esclavo el nodo maestro se encarga de la corrección de errores en estos sin emitir notificación al cliente.

2.2.3 Spark

Apache Spark es una plataforma de computación de código abierto para el procesado de datos a gran escala [7]. Admite varios lenguajes de programación como Java, Scala, Python o R. Para este proyecto se utiliza el lenguaje Java.

Spark mantiene la escalabilidad y la tolerancia a fallos de MapReduce pero añade nuevas funcionalidades para mejorar el rendimiento como el uso de *Resilient Distributed Data* (RDD) para manejar los datos.

La principal diferencia respecto a MapReduce es que Spark almacena los datos leídos en la memoria caché del nodo utilizado, frente a MapReduce que siempre lee los datos del disco duro. Esto provoca que los datos son leídos solo una vez desde HDFS y almacenados en la memoria principal. Ello provoca una reducción respecto al número de veces que se escribe y se lee utilizando *MapReduce*. La lectura en memoria es mucho más rápida que leer y escribir en disco, por lo que utilizar esta solución puede permitir una mayor velocidad en la ejecución de los programas y mejorar su rendimiento.

Está basado en una componente básica llamada Spark *Core,* que contiene las funcionalidades básicas de Spark que son utilizadas por todos sus componentes. Engloba la gestión de la memoria, la recuperación frente a fallos, el *scheduling* de las tareas o la división de las RDDs en particiones y su disposición en los nodos del cluster.

Una *Resilient Distributed Datasets (RDDs)* es el objeto básico en Spark. Se distribuyen en el *cluster* mediante particiones que se envían a los nodos donde se almacenan en la memoria principal siempre que sea posible o sobre el disco local en caso de que no exista suficiente memoria principal disponible. Spark permite que se ejecuten en paralelo, cada nodo corre el código específico en su partición de la RDD.

Una RDD puede crearse de un fichero de texto utilizado como entrada o como salida de una operación realizada sobre otra RDD. Estas RDDs pueden ser almacenadas fácilmente en un archivo HDFS, mediante el método *saveAsTextFile*.

La estructura de un programa en Spark se basa en el *Driver,* el cual contiene el método *main,* crea las RDDs y define las operaciones realizadas por el programa. En él también se incluye el objeto de contexto de Spark (*SparkContext)* que permite la creación de RDDs y el lanzamiento de los ejecutores (*executors)* que en paralelo realizan específicas operaciones sobre las RDDs.

Los nodos esclavo ejecutan la aplicación por medio de *executors,* y cada uno de ellos corre sobre la partición de la RDD especificada en el *driver*.

Spark permite realizar a diferencia de *MapReduce* dos tipos de operaciones sobre sus objetos RDD.

-Las operaciones de transformación, que corresponden con operaciones realizadas sobre una RDD que devuelven una nueva RDD. Estas transformaciones solo son ejecutadas cuando una acción es aplicada en la nueva RDD generada. Un ejemplo de transformaciones corresponde con la operación *filter*, *map* o *union*.

-Las operaciones de acción, que devuelven un resultado al programa *Driver* o escriben el resultado en la salida de almacenamiento HDFS. Ejemplo de acciones son: *collect, count, reduce…*

El contener estas operaciones implementadas facilita la programación y permite reducir el número de líneas de código utilizadas para un programa frente a MapReduce.

Spark también introduce nuevos componentes que permiten su uso para nuevas aplicaciones como son: Spark para datos con estructura SQL, Spark para Streaming a tiempo real, MLlib para minería de datos (data mining) y GraphX para procesado de gráficos.

2.2.4 Hive

Hive es un almacén de datos construido sobre Hadoop. Se basa en una estructura en la cual mediante un lenguaje muy similar a SQL (*Structured Query Language*) se permite la utilización de Hadoop con lo que posibilita la realización de análisis sobre grandes volúmenes de datos almacenados en HDFS sin la necesidad de tener conocimientos de programación en Java.

El lenguaje utilizado es Hive *Query Language* (HQL), en él, las declaraciones realizadas son similares a las utilizadas por el estándar SQL, aunque se encuentra más limitado en los comandos que entiende. Toda la descripción de este lenguaje se encuentra disponible en la página oficial de Apache Hive [8].

Para su funcionamiento en paralelo sobre un *cluster, Hive* transforma automáticamente las declaraciones realizadas en lenguaje HQL en programas ejecutables mediante MapReduce, implementando las funciones *map* y *reducer* necesarias. Estos programas son ejecutados a través del *cluster* de *Hadoop*.

Para trabajar con los datos *Hive* utiliza tablas. Se pueden crear dos tipos de tablas:

- *Managed*: Es el tipo de tabla que utiliza por defecto. En este tipo de tablas, los datos se mueven al directorio (*warehouse*) de *Hive*.
- *External*: En las tablas de este tipo se hace referencia a la localización de los archivos que se encuentran fuera del directorio almacén de *Hive*.

La operación de cargar los datos en una tabla de tipo *managed* es muy rápida ya que consiste simplemente en mover un archivo dentro del propio sistema, no lo copia. *Hive*

no comprueba que los archivos introducidos en la tabla cumplen con el esquema establecido en ella. Para notificar cualquier error o campo que no se corresponda con lo establecido se utiliza la sentencia SELECT que permite comprobar las columnas de la tabla.

Hive ofrece la posibilidad de separar los datos pertenecientes a sus tablas con el uso de particiones. Esto resulta muy útil cuando se quieren diferenciar los archivos de entrada. Por ejemplo, en nuestro caso utilizaremos estas particiones para poder diferenciar el tráfico almacenado por día y hora.

2.3 Tstat

Tstat es una herramienta diseñada para automatizar la recogida de estadísticas sobre agregaciones de tráfico, mediante una monitorización de este. Es un analizador pasivo de tráfico de código abierto que ha sido desarrollada en la universidad Politécnica de Torino por el *Telecommunication Network Group* (TNG).

Para realizar el análisis de los datos *Tstat* monitoriza pasivamente las cabeceras de los paquetes transmitidos por el enlace a niveles de red y superiores y genera registros del tráfico agrupando las conexiones mediante flujos de tráfico. Estos flujos de tráfico se definen utilizando una regla y los paquetes que la verifican son agrupados bajo el mismo *flowID*. En nuestra red se utiliza la siguiente regla para agrupar tráfico del protocolo de control de transmisión (TCP).

FlowID = (Protocolo IP utilizado, Dirección IP origen, Puerto origen, Dirección IP destino, Puerto destino, Número de secuencia).

Por ejemplo, para detectar una nueva conexión TCP, Tstat analiza el tráfico realizando una búsqueda en él hasta encontrar los paquetes que marcan el inicio de una conexión TCP, el paquete SYN. El proceso de establecimiento de una conexión TCP queda definido en el anexo 2. Una vez detectados estos mensajes todo el tráfico perteneciente a esta conexión se almacena en un mismo flujo.

Por tanto, esta herramienta permite diferenciar entre diferentes flujos de tráfico para realizar su posterior análisis.

Tstat no almacena todos los datos pertenecientes al *payload* de un determinado flujo de paquetes ya que generaría tal cantidad de información que no sería posible su uso. En cambio, esta herramienta extrae la información deseada disponible en las cabeceras de los paquetes y almacena los resultados en tablas con los valores prefijados. Estas tablas se crean para permitir diferenciar el tipo de tráfico analizado. Su descripción completa se encuentra facilitada por la universidad creadora [3].

En este trabajo se ha centrado el análisis en la tabla "log_tcp_complete". Esta tabla crea un nuevo flujo de datos cada vez que se detecta una nueva conexión TCP. Cuando Tstat registra el envío de segmentos FIN/ACK o RST o no detecta envío de paquetes por un tiempo de *timeout*, por defecto de 5 minutos desde que se envió el último paquete, el flujo de datos se finaliza.

En esta tabla se almacenan parámetros disponibles en las cabeceras IP y TCP como son las direcciones IP utilizadas, el volumen de tráfico generado en direcciones de subida y bajada, puertos utilizados, número de ACKs enviados... Así como el tiempo en que se registra el envío del primer y último paquete o detectar el tipo de tráfico de nivel superior enviado que es posible analizar mediante las cabeceras de nivel superior.

La obtención de todos los parámetros registrados en la tabla no es siempre posible puesto que Tstat no puede analizar el *payload* de todo tráfico que se encuentre cifrado, como puede ser HTTPS, aunque es posible averiguar que se está utilizando este tipo de conexión analizando el puerto utilizado, en este caso el 443, y el uso de TLS (Transport Layer Security), que se puede obtener del campo *connection type.*

También se utiliza para el análisis la tabla "log_http_complete", esta tabla recoge información perteneciente a conexiones HTTP y HTTPS. Cada entrada en la tabla coincide con el envío de un paquete HTTP *query* o HTTP *request* ya que para este caso se recogen diferentes parámetros en función del tipo de paquete HTTP. En esta tabla se recogen parámetros de la cabecera IP como son las direcciones utilizadas y campos pertenecientes a HTTP como el *Request method* o el *User Agent.*

El funcionamiento del protocolo HTTP requiere del envío de un paquete DNS para realizar la conversión de la dirección URL (*Uniform Resource Location*) solicitada por el usuario en una dirección IP a la cual realizar la conexión mediante el envío de un mensaje utilizando el protocolo DNS. En la cabecera de este paquete DNS se encuentra el dominio que el usuario desea visitar. Tstat también recoge en un campo de la tabla log_http_complete el valor de este campo. Esto puede permitir que incluso de una conexión HTTPS en la cual el *payload* se encuentra cifrado, sea posible obtener el dominio que el usuario desea visitar, ya que estos paquetes DNS viajan en la red sin ningún tipo de cifrado.

En nuestra red esta herramienta es utilizada con el propósito de crear estos flujos de conexión de datos que permitan realizar un análisis del tráfico. Como hemos comentado, Tstat utiliza diferentes tipos de tablas para diferenciar el tráfico en la red, en este caso se generan cinco registros diferentes. Estos registros son creados cada hora, por lo que no es una herramienta que utilicemos para análisis de tráfico en tiempo real, si no, que el análisis siempre se realiza a posteriori, con un tiempo mínimo de una hora.

Por ello, nos centraremos en el análisis sobre datos previamente almacenados y no en streaming sobre tráfico en tiempo real.

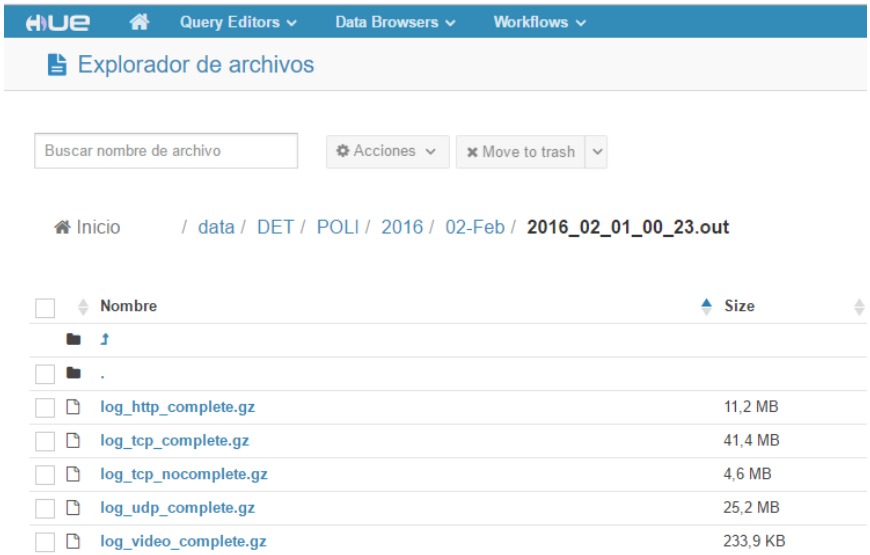


Figura 0-2: Registros de tráfico generados

Los cinco registros de datos generados son:

-log_tcp_complete

-log_http_complete

-log_udp_complete

-log_tcp_nocomplete

-log_video complete

Y como hemos indicado nos centraremos en el estudio de los archivos log_tcp_complete y log_http_complete.

CAPÍTULO 3

Arquitectura del sistema

Este capítulo detalla el escenario sobre el que se realiza el estudio y el hardware utilizado.

3.1 Escenario

La red en la cual se ha realizado la monitorización del tráfico corresponde con un departamento de la Universidad Politécnica de Turín, donde se pueden localizar unos 20000 usuarios.

Este tráfico se encuentra monitorizado mediante la herramienta Tstat, como se explica en el capítulo 2.3. En la red esta herramienta se encuentra situada entre la salida de la red interna de la universidad e Internet. Esta posición le permite monitorizar todo el tráfico que atraviesa la red.

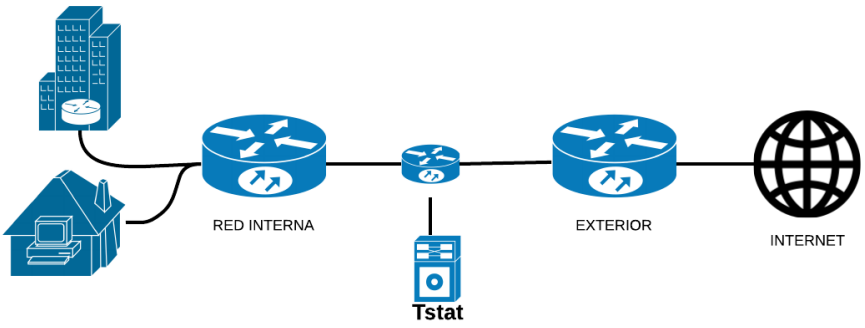


Figura 0-1. Posición de Tstat en la red

Esta herramienta genera *logs* de gran tamaño, para reducir el peso de estos archivos se comprimen utilizando *GZIP* y el resultado se guarda en un repositorio central de HDFS. Ello permite acceder a los datos mediante una conexión a internet y poder utilizar las herramientas de *Big Data* descritas en el capítulo anterior para su análisis.

Este directorio de HDFS se encuentra funcionando sobre un *cluster* instalado en la Universidad Politécnica de Torino, el cual detallaremos brevemente a continuación:

3.2 *Cluster*

El *cluster* consiste en un grupo de nodos organizados en dos racks como detalla la figura 3-2.

Este *cluster* se encuentra formado por:

- 30 nodos trabajadores (*workers*).
- 3 nodos maestro (*master nodes).*
- 4 *switchs* para interconectar todos los nodos
- Y una fuente de energía ininterrumpida utilizada para prevenir posibles caídas de la red eléctrica.

La descripción completa de estos dispositivos se encuentra en el sitio web de *BigData* de la universidad [1].

Estos *workers* permiten obtener una memoria RAM en paralelo superior a los 2TB, 276 discos duros y 768 TB de almacenamiento con una velocidad de lectura de disco de 45 GB/s.

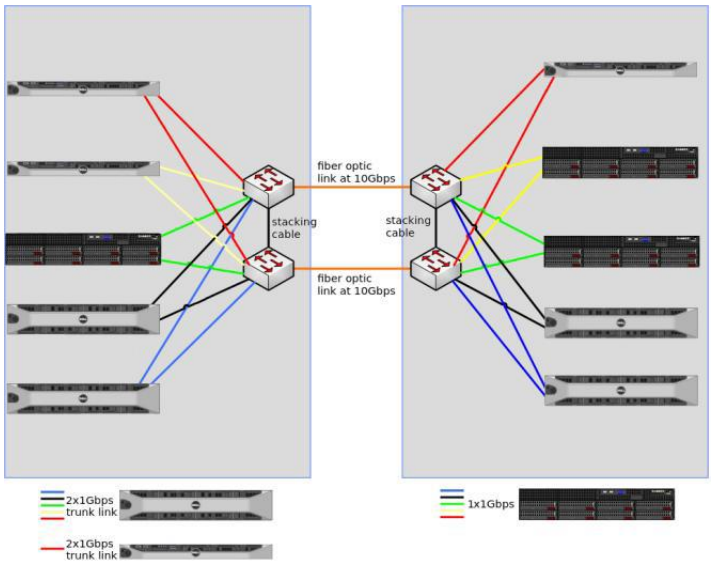


Figura 0-2. Estructura del *cluster*. Fuente: *Idilio Drago,[1]*

Cada máquina se encuentra conectada a dos *switches* dentro de su *rack* para tratar de reducir posibles problemas de congestión en los nodos centrales.

Estas máquinas tienen instalado el sistema operativo *Ubuntu Server LTS* y la plataforma Apache Cloudera en la cual se incluye el software de Apache Hadoop. Se despliegan los siguientes servicios de Hadoop, que se pueden observar mediante Cloudera manager en la figura 3.3.

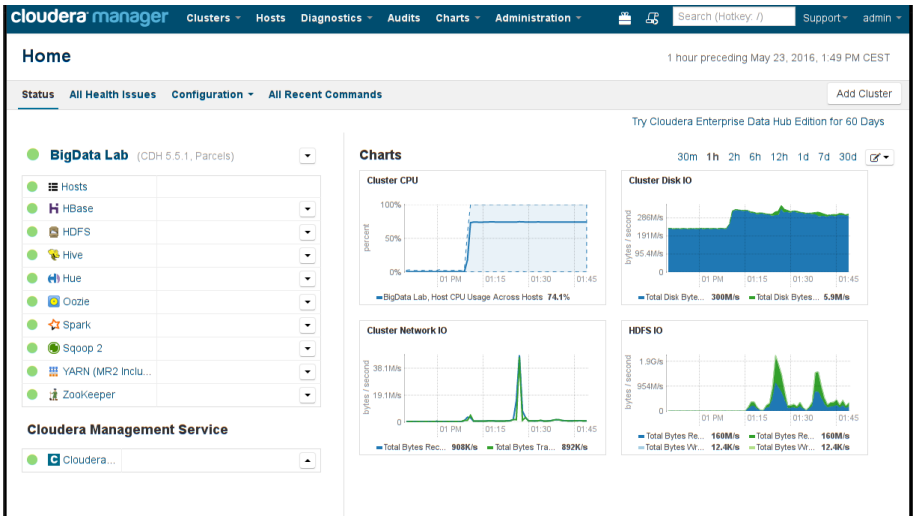


Figura 0-3. Interfaz web de Cloudera Manager

Como hemos indicado se hará uso de los servicios Hive, Spark y MapReduce, así como de la interfaz web proporcionada mediante Hue.

3.3 Metodología

Este apartado describe como se han realizado las medidas y obtenido los resultados.

Para obtener unos valores fiables los resultados sobre un mismo test se han realizado sobre la misma carga del *cluster* para evitar tiempos de espera de los test realizados o sobrecarga en el *cluster*. Esto se ha realizado analizando los registros generados durante la ejecución para observar que no se han producido resultados inesperados como pudiera ser tiempos de espera o una mayor utilización del *cluster* que introduzca retraso en la ejecución del trabajo. En el caso de que se produzca el resultado es descartado.

Los resultados se han obtenido tras realizar la operación en quince ocasiones. Para estimar el valor a utilizar se ha realizado una medida de la distribución de los resultados, descartando el valor máximo y el mínimo obtenido. Estos test se han realizado uno seguido de otro para encontrarse con niveles de carga en el *cluster* similares, obteniendo esto mediante los registros de cada operación.

CAPÍTULO 4

Aplicación de las herramientas de Big Data para el análisis de red

---

En este capítulo se realiza el estudio de las diferentes herramientas de Apache Hadoop utilizadas y el rendimiento obtenido con ellas en su uso para analizar el tráfico almacenado en el *cluster*.

4.1 Preparación del entorno de trabajo

Para poder proceder con el análisis de las herramientas de *Big Data* es necesario preparar el escenario previamente para que sea posible su correcto funcionamiento.

Como se ha explicado en los capítulos 2 y 3, Tstat realiza cada hora un nuevo registro. Este es almacenado en un directorio HDFS organizado por año, mes, día y hora en que se ha producido el tráfico. Así, la figura 2.2 muestra los *logs* creados en día 1 de febrero de 2016 a las 00:00 durante una hora. La existencia de diferentes *logs* en un mismo directorio va a derivar en un problema si intentamos ejecutar un programa sobre ellos puesto que *MapReduce, Hiv*e y *Spark* no permiten especificar el nombre del archivo que se quiere ejecutar, si no que se debe especificar el directorio a ejecutar y al contener estos varios archivos con diferente estructura no se podrían analizar los datos con estos métodos.

Para evitar esto se desarrolla el script "copy.sh" adjuntado en el anexo 1, el cual copia directamente los archivos que se desean analizar en nuevos directorios, de la forma explicada en el anexo.

Para poder realizar un análisis de esos datos con la herramienta *Hive* primero se debe proceder a la creación de tablas y la carga de estos datos en ellas. Se consigue realizar este procedimiento de manera automática mediante el desarrollo de scripts adjuntos en el anexo 1.

Cada tipo de *log* necesita una nueva tabla Hive, ya que en cada registro se almacenan diferentes datos y el archivo tiene una estructura diferente. La estructura de cada log se encuentra definida en la primera línea de estos. Utilizando esta línea se consigue automatizar el proceso de creación de una tabla a partir de cualquier de *log*, sin importar el tipo de este. Por ejemplo, la primera fila del archivo log_tcp_complete la podemos observar en la figura 4.1. Esta fila indica el nombre de cada columna y su posición en el

fichero. También se han creado diferentes particiones para mejorar la organización de las tablas que permiten separar el tráfico diferenciando por día y hora.

```
#15#c_ip:1 c_port:2 c_pkts_all:3 c_rst_cnt:4 c_ack_cnt:5 c_ack_cnt_p:6 c_bytes_uniq:7 c_pkts_data:8 c_bytes_all:9 c_pkts_retx:10 c_bytes_retx:11 c_pkts_o
oo:12 c_syn_cnt:13 c_fin_cnt:14 s_ip:15 s_port:16 s_pkts_all:17 s_rst_cnt:18 s_ack_cnt:19 s_ack_cnt_p:20 s_bytes_uniq:21 s_pkts_data:22 s_bytes_all:23 s_
pkts_retx:24 s_bytes_retx:25 s_pkts_ooo:26 s_syn_cnt:27 s_fin_cnt:28 first:29 last:30 durat:31 c_first:32 s_first:33 c_last:34 s_last:35 c_first_ack:36 s
_first_ack:37 c_isint:38 s_isint:39 c_iscrypto:40 s_iscrypto:41 con_t:42 p2p_t:43 http_t:44 c_rtt_avg:45 c_rtt_min:46 c_rtt_max:47 c_rtt_std:48 c_rtt_cn
t:49 c_ttl_min:50 c_ttl_max:51 s_rtt_avg:52 s_rtt_min:53 s_rtt_max:54 s_rtt_std:55 s_rtt_cnt:56 s_ttl_min:57 s_ttl_max:58 p2p_st:59 ed2k_data:60 ed2k_si
g:61 ed2k_c2s:62 ed2k_c2c:63 ed2k_chat:64 c_f1323_opt:65 c_tm_opt:66 c_win_scl:67 c_sack_opt:68 c_sack_cnt:69 c_mss:70 c_mss_max:71 c_mss_min:72 c_win_ma
x:73 c_win_min:74 c_win_0:75 c_cwin_max:76 c_cwin_min:77 c_cwin_ini:78 c_pkts_rto:79 c_pkts_fs:80 c_pkts_reor:81 c_pkts_dup:82 c_pkts_unk:83 c_pkts_fc:84
 c_pkts_unrto:85 c_pkts_unfs:86 c_syn_retx:87 s_f1323_opt:88 s_tm_opt:89 s_win_scl:90 s_sack_opt:91 s_sack_cnt:92 s_mss:93 s_mss_max:94 s_mss_min:95 s_wi
n_max:96 s_win_min:97 s_win_0:98 s_cwin_max:99 s_cwin_min:100 s_cwin_ini:101 s_pkts_rto:102 s_pkts_fs:103 s_pkts_reor:104 s_pkts_dup:105 s_pkts_unk:106 s
_pkts_fc:107 s_pkts_unrto:108 s_pkts_unfs:109 s_syn_retx:110 http_req_cnt:111 http_res_cnt:112 http_res:113 c_pkts_push:114 s_pkts_push:115 c_tls_SNI:116
 s_tls_SCN:117 c_npnalpn:118 s_npnalpn:119 c_tls_sesid:120 c_last_handshakeT:121 s_last_handshakeT:122 c_appdataT:123 s_appdataT:124 c_appdataB:125 s_app
dataB:126 fqdn:127 dns_rslv:128 req_tm:129 res_tm:130
```

Figura 0-1. Primera fila del *log log_tcp_complete* que incluye la estructura de este.

La creación de una tabla *Hive* exige recibir como parámetros de cada columna un nombre y el tipo de dato que representa (*integer, string, long…)* [8]. Como esta información no se recibe del *log* capturado, se genera previamente un archivo con todos los nombres de columna utilizados en los *logs* y su respectivo tipo de dato. Para ello se crea un *script* que permite la creación de esta tabla. Este script se encuentra en el anexo 1 llamado "create_table.sh". Su funcionamiento se explica en el anexo.

Una vez se ha creados la tablas se procede a automatizar la carga de los datos en ellas. Para realizar este proceso también se hace uso de un *script* que permite introducir nuestros datos en las respectivas tablas. Se crean dos *scripts* para este proceso, el primero, load_partition.sh, permite cargar datos en una tabla de tipo *managed* y el segundo, partition.sh, permite el mismo proceso para tablas tipo *external.* Ambos *scripts* se encuentran en el anexo 1.

Para cargar múltiples ficheros en Hive se requiere de varias *queries*, una por *log.* Como buscamos optimizar el tiempo de ejecución de este proceso se accede a *Hive* en modo no iterativo mediante línea de comandos. En este modo se carga directamente un script con todas las *queries* a utilizar y se consigue reducir el tiempo total de la operación, puesto que se realiza una única conexión. Si se utilizara en un modo iterativo se realizaría una conexión cada vez que se pide ejecutar una nueva *query* y este tiempo suele ser un tiempo bastante alto (varios segundos), en comparación por ejemplo, al tiempo que pueda tardar cargar un archivo en la tabla (menos de un segundo).

Realizando un análisis de los tiempos de carga de los archivos en la tabla podemos comprobar algunas propiedades de estas. Al realizar la carga en una tabla de tipo *external* comprobamos como no existe diferencia en cuanto a vincularla a datos de mayor o de

menor tamaño puesto que *Hive* no altera la localización de estos archivos, solo crea una referencia a la localización, que ya se encuentra en el repositorio HDFS. En cambio, en una tabla de tipo *managed* los archivos se mueven al directorio de Hive, el procedimiento de mover un archivo no tiene coste computacional. Por lo tanto, tampoco se encuentra diferencia en el tiempo de carga de un archivo para diferentes tamaños de archivo.

Tamaño de archivo	Hive *external table*	Hive *managed table*
36 Mbyte	0.105 s	0.649 s
180.3 Mbyte	0.102 s	0.629 s
390 Mbyte	0.110 s	0.683 s

Tabla 0-1: Tiempo necesario para cargar un archivo en Hive

Cuando nos encontramos con múltiples archivos, al realizar la conexión de manera no iterativa conseguimos que el tiempo total de crear las tablas sea el número de archivos utilizados por el tiempo para crear uno, todos los archivos tardan lo mismo a cargarse.

Este resultado es importante para Hive puesto que es necesario que se carguen todos los datos a utilizar en su base de datos antes de poder proceder a su estudio. Spark y MapReduce también necesitan que los datos se encuentren almacenados en un repositorio de HDFS para su utilización aunque en estos casos los datos no necesitan ser cargados en tablas como en Hive. Como los datos que utilizamos se encuentran almacenados en HDFS este resultado se muestra de interes únicamente para Hive.

4.2 Comparación del tiempo de ejecución para las diferentes herramientas (*Benchmark*)

En este apartado se realizan una serie de análisis sobre los *logs* utilizando las herramientas estudiadas y comparando los resultados obtenidos con cada una. Para conseguir el resultado más preciso posible se realiza un análisis estadístico del tiempo obtenido, al realizar la misma petición en quince ocasiones descartando el mayor y el menor resultado del tiempo de ejecución.

Se pretende optimizar este tiempo mediante la realización del código y el uso de la estructura disponible en el *cluster*. El número de *mappers* que se puede utilizar viene determinado por el número de ficheros a analizar, esto ocurre así debido a la compresión que se ha utilizado para almacenar estos *logs*. Los programas utilizados permiten utilizar directamente como parámetro de entrada un fichero comprimido mediante GZIP. Este formato utilizado para comprimir no permite trabajar con el fichero dividido sin su previa

descompresión. Debido a ello, estos ficheros se almacenan en HDFS forzando un único bloque de datos por fichero. Esto provoca que sea necesario el uso de un *mapper* por *log* y no se pueda modificar este número. Esto no es un problema para ejecutar los programas en paralelo puesto que nos encontramos en un escenario en el que tenemos una alta cantidad de ficheros de un tamaño reducido, centenares de MegaBytes. Siempre que se encuentren suficientes nodos disponibles esto no es una limitación.

Para el primer análisis realizado se consiguen obtener los puertos más utilizados por las conexiones TCP. El número de puerto utilizado lo encontramos en el fichero log_tcp_complete en la columna s_port, para su obtención procesamos por separado los datos pertenecientes a un día de tráfico y a un mes.

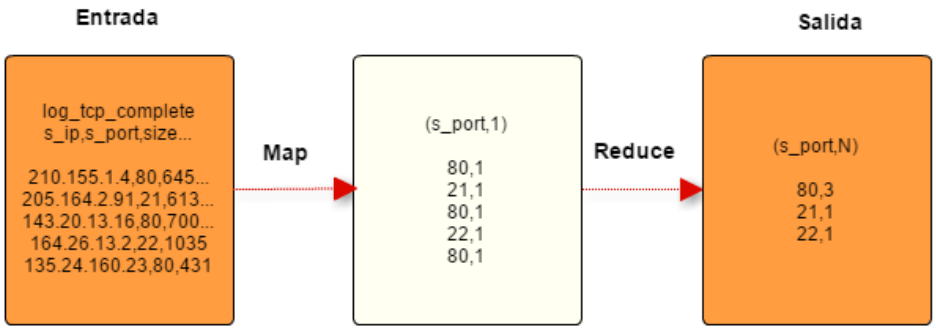


Figura 0-2: Flujo de datos *Map-Reduce*.

Para obtener estos resultados se procesan los datos correspondientes a un día de tráfico y a un mes de manera separada. Los análisis se han realizado para diferentes tamaños de entrada distinguiendo entre un día y un mes. El tamaño de un día de trafico almacenado es de 4.5 Gigabytes frente al tamaño de un mes de datos 104 Gigabytes y el número de ficheros utilizados son 24 para un día y 720 para un mes.

Tiempo ejecución	1 día de tráfico (s)	1 Mes de tráfico (s)
Hive *External*	108	170
Hive *Managed*	106	168
SPARK	46	72
Map Reduce	49	78

Tabla 0-2: Tiempo de ejecución, para obtener los puertos TCP más usados

En este caso se puede observar de la tabla 4.2, como el tiempo de ejecución del programa con la herramienta Hive es dos veces el obtenido mediante MapReduce. En este caso, esto se produce debido a que Hive separa el programa a realizar en dos operaciones de MapReduce automáticamente al transformar el código HSQL. En cambio al realizar esta operación mediante MapReduce se puede implementar el programa de tal manera que permita obtener la misma solución en una única ejecución, por medio de un *mapper* y un *reducer*.

El resultado no muestra diferencia entre la ejecución en Hive utilizando tablas de tipo externo o *managed*, como se esperaba ya que ambas son referencias a archivos almacenados en HDFS.

En cuanto a la diferencia de utilizar el tráfico perteneciente a un único día o un mes de datos, el uso de un volumen de tráfico 30 veces superior genera una diferencia baja en los tiempos de ejecución, cercana al doble de tiempo utilizado. Aquí reside la ventaja del uso del *cluster* y el trabajo en paralelo que realizan sus nodos, permitiendo trabajar con grandes volúmenes de datos.

En el siguiente análisis realizado se obtiene el volumen de tráfico TCP que circula por la red, es decir, el tráfico entrante y saliente a la red. Estos datos los encontramos en las columnas c_bytes_uniq y s_bytes_uniq del fichero log_tcp_complete.

Como parte de los resultados cabe destacar la longitud de los programas realizados mediante la herramienta Hive. Mediante una única línea de código HSQL se puede obtener el mismo resultado que con la herramienta MapReduce en la cuál es necesario implementar cuatro clases Java diferentes.

Para comprobar las diferencias se realizan análisis sobre el tráfico de un único día y el perteneciente a un mes. Para este trabajo se utiliza un único *reducer* y se devuelven dos valores, el tráfico total de entrada y de salida. Se hace uso de un *combiner* a la salida del *mapper* para reducir los datos enviados a la red.

Tiempo ejecución	1 día de tráfico (s)	1 Mes de tráfico (s)
Hive *External*	72	97
Hive *Managed*	62	98
Spark	65	102
MapReduce	155	240

Tabla 0-3: Tiempo de ejecución, para obtener el tráfico TCP total.

Hive utiliza la operación *Sum* para obtener los valores del tráfico total. Esta operación obtiene buenos resultados en el tiempo de operación al ser la herramienta más rápida junto con Spark en conseguir el resultado. Nuevamente los resultados son análogos para tablas de tipo *external* e *internal.* Aparece una diferencia entre el resultado obtenido para el caso anterior ya que ahora obtiene un resultado mucho mejor. Esto es debido a la diferencia en el número de trabajos MapReduce ejecutados, para este caso utiliza los mismos que las otras herramientas mientras que para el caso anterior utilizaba el doble.

En cambio, MapReduce obtiene los peores resultados durante el análisis. La herramienta no implementa una salida en la cual tenga que proporcionar dos valores como es el caso de Spark o Hive y se debe realizar mediante el uso de una nueva clase de tipo *Writable* que obtiene peores resultados que las otras.

Tras este análisis se realiza un nuevo programa para poder obtener la cantidad de tráfico generado por los cincuenta sitios web más visitados. El tráfico web corresponde con tráfico sobre HTTP o HTTPS, para diferenciar este tipo de tráfico en el fichero log_tcp_complete se hace uso de la información disponible en la columna con_t y s_port; que corresponden con los valores del tipo de conexión y el puerto utilizado en el servidor respectivamente. Las conexiones HTTP se corresponden con un valor del campo con_t de 1 y las conexiones HTTPS están marcadas con un valor del campo con_t de 8192, utilizado para todas las conexiones TLS, sobre el puerto 443, el puerto utilizado por HTTPS.

Este programa es más complejo que el anterior, puesto que primero se debe filtrar por tráfico HTTP y obtener los dominios web más visitados, calculando el tráfico que generan.

La columna fqdn proporciona el valor del fully qualified domain name que proporciona el nombre del dominio en las tres jerarquías de DNS: nombre del host, subdominio y el dominio de nivel superior. Con esta columna obtenemos los dominios que son visitados en la red.

Tiempo ejecución	1 día de tráfico (s)	1 Mes de tráfico (s)
Hive *External*	114	212
Hive *Managed*	116	221
Spark	135	280

Tabla 0-4: Tiempo de ejecución, para obtener el tráfico generado por los sitios web más visitados.

Tiempo ejecución	1 *Reducer* (s)	2 *Reducer* (s)	8 *Reducer* (s)	80 *Reducer* (s)	140 *Reducer* (s)
MapReduce	1250	901	778	840	850

Tabla 0-5: Tiempo de ejecución, para obtener el tráfico generado por los sitios web más visitados, variando el número de *reducers*, para un mes de tráfico.

En este caso se modifica el número de *reducers* utilizados en Java MapReduce para analizar la diferencia. El uso de un mayor número de estos permite realizar más trabajos en paralelo. Esto funciona mejor cuanto mayor es el volumen de resultado extraído en la etapa *mapper*. Para este caso observamos como el tiempo de análisis se reduce al aumentar el número de *reducers* de uno a dos. Al aumentar el valor de los *reducers* a valores muy altos no se obtiene ninguna ganancia puesto que algunos no están recibiendo datos y generan una salida nula. En este caso el mejor tiempo se ha obtenido al utilizar 8 *reducers*, este valor ha sido averiguado mediante un barrido del número de *reducers* utilizado.

Aun así, los resultados de Java MapReduce son peores que los obtenidos mediante las otras herramientas. En este caso, Hive consigue el menor tiempo de ejecución del sistema.

El siguiente análisis se realiza sobre el archivo log_http_complete, en el cual se almacenan parámetros pertenecientes al protocolo HTTP. Para este análisis se hace uso de la operación *join* disponible tanto en Hive como en Spark. No se utiliza la herramienta MapReduce puesto que esta no implementa la operación *join*. Este programa se ha implementado para conseguir filtrar por un dominio DNS utilizado por un servicio web, como google.com, y obtener las direcciones IP utilizadas por sus servidores, este resultado se une mediante la operación *join* con el fichero completo en el que se encuentran todas las direcciones IP y en el caso de coincidir se extrae el valor del dominio visitado si no corresponde con el especificado en el programa. Esto permite obtener que servicios operan sobre las mismas direcciones IP.

El tamaño de un día de tráfico almacenado correspondiente al fichero log_http_complete corresponde con 3.9 Gigabytes.

Tiempo ejecución	1 día de tráfico (s)
Hive	472
Spark	63

Tabla 0-6 : Tiempo de ejecución, para obtener las direcciones IP que comparte un dominio.

En este caso, Spark obtiene realmente una velocidad de ejecución mucho mayor que Hive. Esto es así puesto que al realizar la lectura del fichero una vez y almacenarla en memoria se puede reducir el tiempo de ejecución cuando el fichero es utilizado en más de una ocasión en el mismo programa. *Hive* en cambio, realiza la lectura en disco de su tabla completa cada vez que necesita utilizarla. En este análisis de mayor complejidad, no es recomendable el uso de la herramienta Hive puesto que los tiempos de análisis aumentan por un factor 8.

Como conclusión de los análisis no podemos concretar una herramienta que obtenga los mejores resultados para todos los casos analizados. Spark obtiene mejores resultados que MapReduce para todos los casos estudiados. Esto es así debido a las mejoras que esta herramienta introduce respecto a MapReduce al implementar nuevas operaciones y una gestión de los datos en memoria que permite reducir el tiempo empleado. La diferencia temporal de esta herramienta respecto a las otras se nota al realizar trabajos de una mayor complejidad en los cuales el almacenamiento de los datos en memoria supone una ventaja.

Por el contrario, MapReduce ofrece los peores resultados de las tres herramientas ya que no permite realizar operaciones diferentes al mapeo y reducción ni consigue un código simple de implementar.

En cuanto a la herramienta Hive cabe destacar la facilidad disponible de realizar los programas directamente con código HSQL. Este modo de programación es mucho más simple que los anteriores pero permite obtener buenos resultados. La mayor ventaja de esta herramienta es permitir a usuarios utilizar herramientas de BigData sin tener conocimientos de lenguajes de programación diferentes a SQL. Esto supone una gran facilidad para que personal con conocimiento de gestión de bases de datos pueda utilizar los beneficios generados por las herramientas de Big Data. También cabe destacar como puede conseguir los mejores resultados para los análisis sobre algunos trabajos más simples.

CAPÍTULO 5

Análisis del tráfico

En esta sección se realiza una introducción acerca de cómo se realizan las mediciones sobre Internet y los resultados obtenidos en la red analizada.

5.1 Caracterización de la red

En este apartado se utiliza la herramienta Spark, puesto que contiene mayor número de funcionalidades, para analizar el comportamiento de la red.

Para caracterizar el comportamiento del tráfico sobre la red se realizan algunos análisis similares a los que se enfrentan para realizar mediciones sobre Internet.

El primer parámetro estudiado es el volumen total de tráfico TCP generado en ella. Estos valores se pueden obtener mediante el fichero log_tcp_complete creado. La figura 5.1 muestra el tráfico total obtenido el día 10 de febrero de 2016. En ella se aprecia cómo se distribuye el tráfico durante un día, concentrándose la mayor cantidad de éste durante las horas centrales. Este resultado es coherente con el esperado por una red perteneciente a la universidad en la cual el tráfico se genera durante las horas en las que tanto alumnos como profesores se encuentran allí.

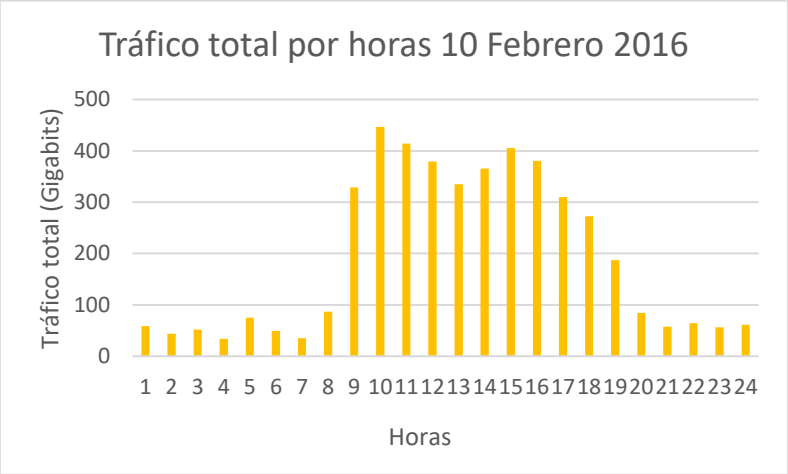


Figura 0-1: Tráfico total en Gigabits del día 10 de Febrero

La imagen superior ilustra cómo se distribuye el tráfico en la red. El volumen de tráfico en las horas centrales es superior a los 100 Megabit/s y se mantiene un tráfico estable durante la noche con tasas cercanas a los 15 Megabit/s.

En la figura 5.2 se muestra el tráfico generado en un mes. Al analizar los resultados se pueden distinguir los días de la semana a los que pertenece cada registro de tráfico. En cada semana se aprecian cinco días en los que el tráfico es mayor y dos con un volumen de tráfico claramente inferior. Los días con mayor tráfico corresponden con días lectivos en los que al encontrarse un mayor número de personas en la universidad el tráfico es mayor.

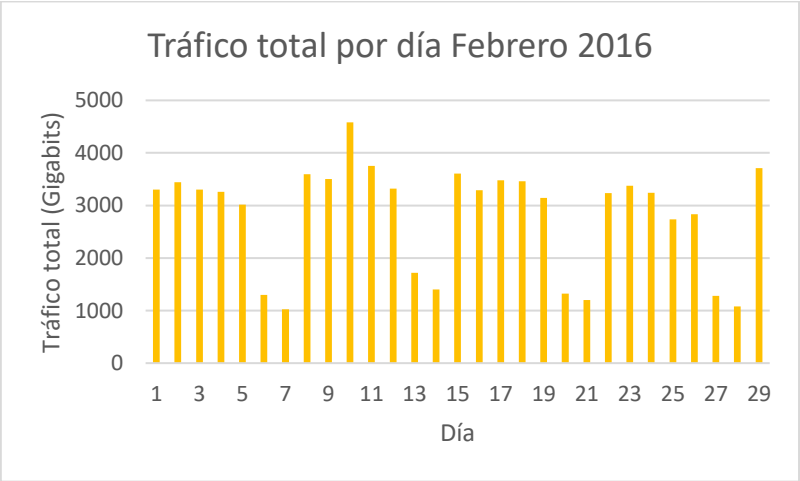


Figura 0-2 Tráfico total en Gigabits durante el mes de Febrero

Para caracterizar este tráfico es normal estudiar los diferentes tipos de tráfico en uso sobre los niveles superiores a TCP/IP, esto muestra los servicios que se están utilizando sobre la red. En este análisis se hace del campo *connection type* que permite distinguir entre diversos servicios utilizados.

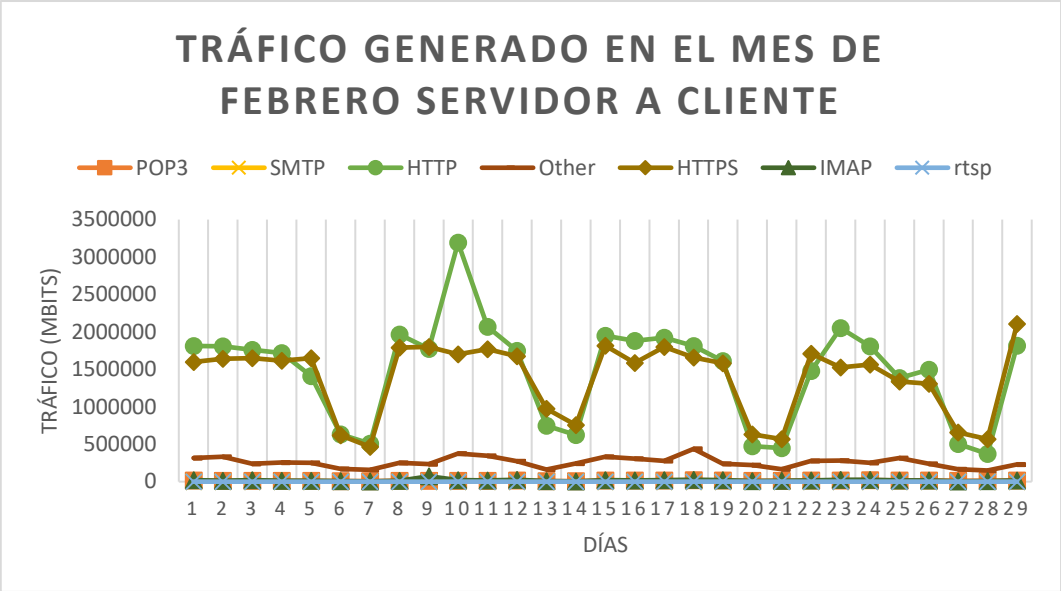


Figura 0-3 : Tráfico entrante a la red diferenciado por el protocolo utilizado durante el mes de febrero.

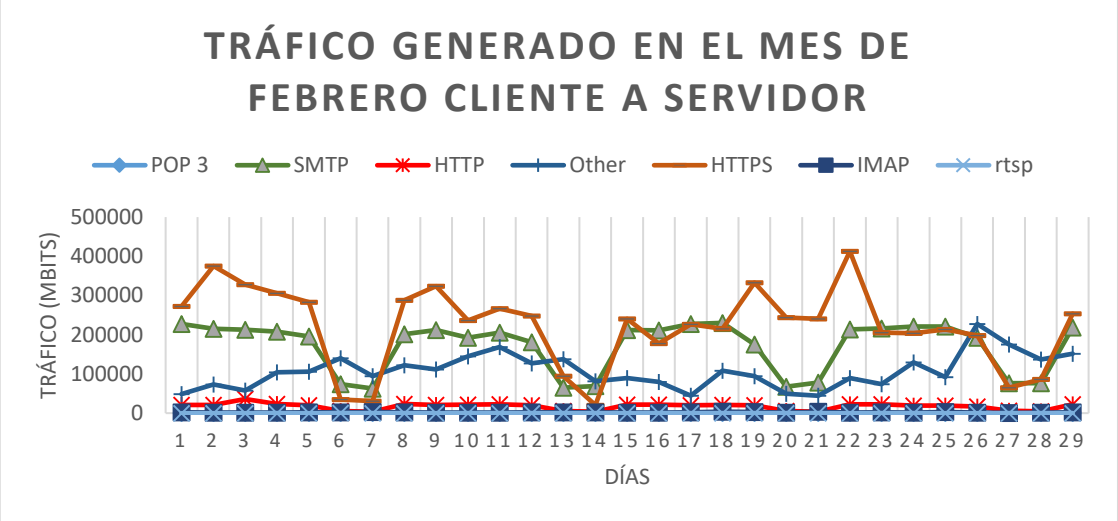


Figura 0-4: Tráfico saliente a la red diferenciado por el protocolo utilizado durante el mes de febrero.

Los resultados obtenidos para el tráfico de entrada muestran como casi todo el volumen de tráfico recibido (93 %), se corresponde con tráfico HTTP y HTTPS, como hemos comentado, estos dos protocolos se utilizan para el acceso a páginas *web*. Estos valores son muy superiores a todo el otro tráfico puesto que la mayor parte de los accesos a servidores se realiza sobre este protocolo. En el gráfico se observa cómo se genera prácticamente el mismo nivel de tráfico HTTP y HTPPS. Cada vez más servicios implementan seguridad en su protocolo de acceso y con ello aumentan el volumen de tráfico HTTPS. Es normal que el uso de este protocolo ascienda puesto que este incorpora encriptación, autenticación e integridad de datos y una mayor privacidad al usuario.

Por tanto, esto supone una limitación a la hora de caracterizar el tráfico correspondiente a diferentes servicios web, puesto que al ir encriptado no se pueden analizar los campos de *payload* relativos a este. Esto dificulta un estudio como el realizado en este trabajo, pero aun así, este tráfico web se podría identificar mediante las direcciones IP utilizadas y las consultas de DNS. Estos valores se encuentran disponibles en la cabecera IP y en el mensaje perteneciente al protocolo DNS que actualmente no viaja cifrado por la red. Con los datos disponibles en el protocolo DNS se pueden obtener los dominios web que se visitan mediante HTPPS.

El resto de tráfico registra un volumen muy bajo de tráfico de entrada frente al tráfico *web*.

Entre el tráfico de salida y de entrada de la red se aprecia una gran diferencia en cuanto al tipo de tráfico utilizado. Aquí el volumen de datos HTTP es bajo puesto que estos paquetes son generalmente pequeños cuando pertenecen a un cliente. La mayoría de estos paquetes HTTP utiliza el método *GET* para pedir información al servidor sobre una determinada página web de la que obtiene en la respuesta el contenido requerido.

Aparece en el tráfico de salida un alto volumen de tráfico generado por el protocolo SMTP (*Simple Mail Transfer Protocol)* e IMAP (Internet Message Access Protocol). Estos protocolos son utilizados para el envío de mensajes de correo electrónico.

Para conocer las aplicaciones que se encuentran en uso sobre la red se realiza un análisis que muestra los puertos utilizados por los clientes. El estudio de los puertos más utilizados por el cliente es interesante debido a la lista desarrollada por la IANA (*Internet Assigned Numbers Authority*) que muestra los puertos conocidos que se relacionan con un servicio [16].

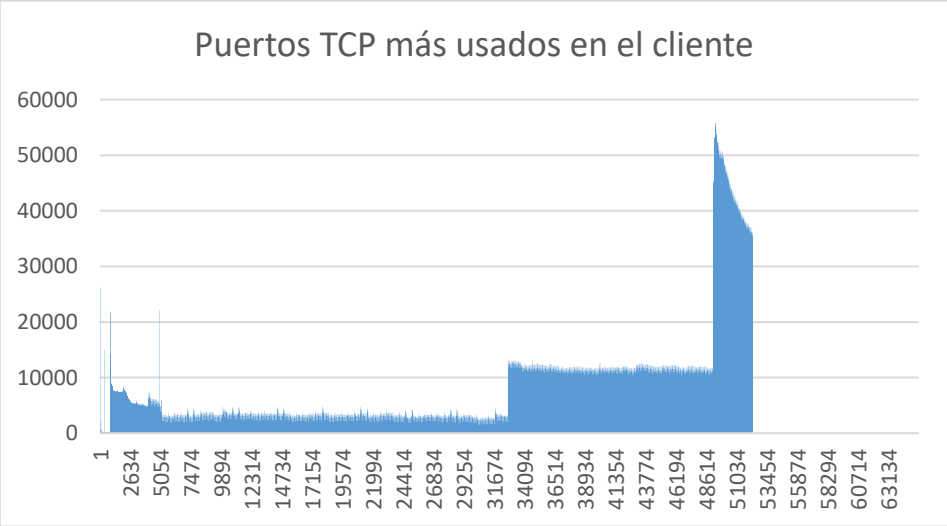


Figura 0-5: Puertos TCP utilizados en el cliente.

En nuestra red, aparecen numerosas entradas sobre los puertos relacionados con IMAP y SMTP, 995 y 445. Esto ocurre porque en la red se encuentra alojado un servidor de

correo electrónico. También aparece un pico en el puerto 443 utilizado por servidores HTTP que implementan seguridad, HTTPS. En otro de los puertos conocidos, el puerto 20, utilizado para conexiones FTP (*File Transfer Protocol*), también encontramos un pico. Este protocolo se utiliza para la transferencia de archivos.

La mayoría de las conexiones salientes se realizan en los puertos "no conocidos" estos puertos son generalmente los que tienen un número superior a 1024. Esto se debe a que son conexiones originadas por clientes en los cuales TCP abre un puerto de origen aleatorio para establecer la conexión.

Los análisis realizados para caracterizar el tráfico también intentan obtener el número de usuarios que actualmente se encuentran conectados a la red. Para redes de un tamaño menor es posible obtener este valor únicamente con el número de direcciones IP en uso en la red. Para redes de un tamaño mayor, una aproximación del número de usuarios se puede realizar mediante el análisis de las cabeceras del protocolo HTTP obteniendo los valores de la firma que genera el navegador mediante campos como *User Agent, Language, Browser Plugin Details.* Esto puede proporcionar una firma única en una red y con ello determinar el número de usuarios.

En nuestra red es complicado establecer un número aproximado de éstos puesto que en el tráfico recogido las direcciones IP almacenadas corresponden a IP públicas tras las que se sitúa una red NAT (*Network Address Translation)* privada y no se dispone de más información de estos para mantener su privacidad. El estudio mediante los campos HTTP tampoco es posible realizarlo puesto que no se registra tanta información de las cabeceras.

Estos tipos de análisis realizados son utilizados para seguridad en redes buscando en ellos patrones de tráfico extraños o intentos de conexiones masivas a servicios pertenecientes a la red entre otros.

5.2 Caracterización de los servicios más utilizados

Como se dispone de un gran número de usuarios, se busca realizar un análisis para obtener los servicios más utilizados en la red. Al ser la mayoría del tráfico almacenado perteneciente a los protocolos de internet HTTP y HTTPS se realiza un estudio sobre los servicios más utilizados bajo estos protocolos.

Primero se establece una lista de los dominios más visitados, para ello se hace uso del campo de *fully qualified domain name* disponible en el registro log_tcp_complete. Los dominios de segundo orden más populares de la red corresponden a: google.com, facebook.com, youtube.com, whatsapp.net, instagram.com, netflix.com y amazon.com

Todos estos servicios utilizan el protocolo HTTPS para su funcionamiento, por lo que hemos comentado en el apartado anterior, será más difícil caracterizar este tipo de tráfico. Por tanto, se busca analizar la relación existente entre direcciones IP y dominios visitados.

Este análisis se realiza sobre algunos de los dominios obtenidos mediante la representación gráfica de la correspondencia entre direcciones IP y dominios. Por ejemplo, al analizar los dominios obtenidos mediante la búsqueda de correspondencias con el servicio web Instagram, aparecen varios dominios de segundo orden: instagram.com, fcbcdn.net, akamai.net y cdninstagram.com. Aparecen dos dominios de Instagram como instagram.com y cdninstagram.com y otros dos pertenecientes a una CDN (*Content Delivery Network*), como es la CDN de Facebook; ya que Instagram pertenece a Facebook; y Akamai. Una CDN es una red de distribución global de contenidos desplegados en múltiples centros de datos. Estas CDN se utilizan para proveer del contenido web a servicios reduciendo el retardo al localizarse en varias posiciones más cercanas al usuario y otorgar de una alta disponibilidad a estos servicios al repartir el tráfico entre los centros de datos. El uso de estas CDN dificulta poder diferenciar entre los diferentes servicios que corren en ellas al no disponer de toda la información sobre el nombre del host de las queries HTTP realizadas por el cliente.

Se representa la asociación para el dominio de segundo orden Instagram.com de las direcciones IP utilizadas frente a los dominios resultantes, representando las direcciones IP como nodos en verde, asociándolas con el nombre completo de *host* DNS (*hostname)* al que realiza la petición.

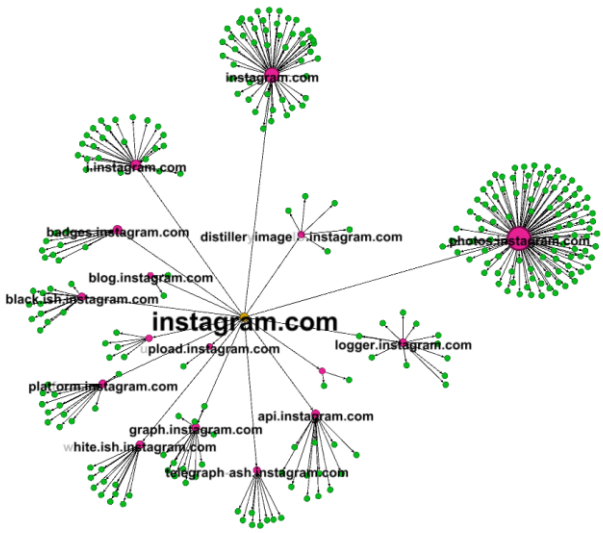


Figura 0-6: Direcciones IP y nombres de *host* para instagram.com

Se observa como para este servicio la mayoría de las direcciones IP utilizadas se encuentran en el dominio perteneciente a photos.instagram.com. Esto indica como este

servicio reparte la mayoría de sus direcciones para la funcionalidad que más tráfico genera de su aplicación, la visualización de imágenes.

Este mismo análisis se repite para el servicio WhatsApp en el cual aparecen dos subdominios de segundo orden: whatsapp.net y whatsapp.com. En este caso no aparece ninguna CDN y al realizar un análisis sobre las direcciones IP obtenidas y los dominios que aparecen se extrae que estas direcciones se utilizan únicamente para el servicio de WhatsApp. Dentro del dominio whatsapp.com aparecen todas las conexiones relativas al servicio WhatsAppWeb, mientras que para el dominio whatsapp.net se obtienen las conexiones realizadas por la versión móvil. La figura 5.7 muestra los subdominios que aparecen.

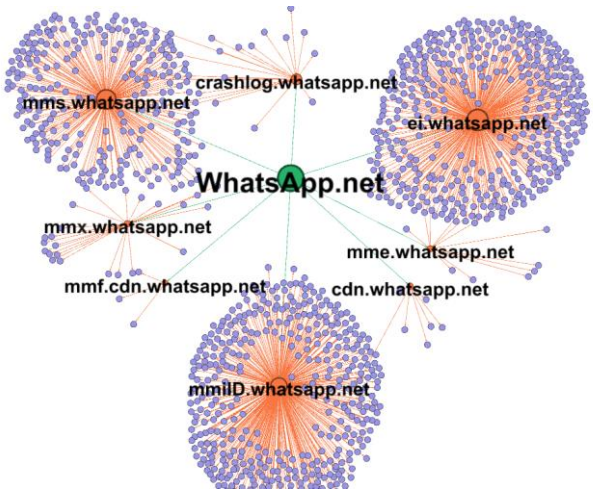


Figura 0-7: Direcciones IP y nombres de *host* para whatsapp.net

En este servicio las direcciones IP se distribuyen principalmente entre tres dominios host DNS.

La siguiente caracterización realizada sobre estos servicios es si las direcciones IP utilizadas corresponden a un único nombre de host. Para ello se extraen todas las direcciones IP utilizadas por el servicio y se comprueba si son utilizadas para más de un servicio web.

Servicio Web	**Instagram.com**	**WhatsApp.net**
Número de direcciones	114	730
Direcciones únicas	67	726

Tabla 0-1 Número total de direcciones IP y el número total de direcciones utilizadas únicamente por el servicio.

Los resultados obtenidos muestran como las direcciones pertenecientes a la red de whatsapp.net se corresponden únicamente con direcciones del propio servicio web. Ya habíamos visto como realizando el filtrado de los dominios web para el servicio de WhatsApp solamente obteníamos resultados correspondientes a whatsapp.net y whatsapp.com, lo que corresponde con lo conseguido.

Por el contrario, para otro servicio como Instagram.com, en el cual hemos observado su relación con Akamai y Facebook, se obtienen resultados muy diferentes. En este servicio casi la mitad de las direcciones utilizadas se comparten para su uso en otros servicios. Es difícil caracterizar exactamente estos servicios y como se encuentran las direcciones compartidas puesto que al encontrarse dentro de un CDN y no disponer de toda la información, no se puede distinguir entre los diferentes servicios.

CAPÍTULO 6

Conclusiones y líneas futuras

Este capítulo presenta una breve descripción resumida y una valoración crítica del conjunto del trabajo realizado, además de proponer posibles líneas futuras.

6.1 Conclusiones

El aumento de la cantidad de tráfico de red ha provocado que para análisis realizado sobre este se tengan que utilizar capaces de trabajar con una cantidad mayor de información, como son las tecnologías pertenecientes a *Big Data*. Con las herramientas estudiadas en este TFG se permite lograr el análisis de grandes cantidades de tráfico de red.

Para llevar a cabo este proyecto se ha partido de una red en la cual mediante la herramienta Tstat se realizan registros del tráfico que la atraviesa. Mediante las herramientas estudiadas de BigData (Hive, Spark y MapReduce) se han realizado análisis sobre este tráfico. Realizando una serie de análisis se han caracterizado las ventajas e inconvenientes relativas a cada una de estas herramientas. Los resultados obtenidos han permitido destacar como se obtienen los mejores resultados mediante la herramienta Spark y como el uso del *cluster* permite trabajar con grandes volúmenes de datos y reducir el tiempo de procesado de los programas de análisis.

El capítulo cinco ha introducido algunas de los análisis y problemas que se realizan para caracterizar el tráfico de red. En la red estudiada se ha obtenido un resultado que muestra como casi todo el tráfico TCP generado pertenece a HTTP y HTTPS.

También se ha realizado un análisis sobre la relación existente entre un dominio y las direcciones IP, mediante la visualización de la asociación entre dominio y direcciones IP.

6.2 Líneas futuras

A la vista de los resultados obtenidos cabe la posibilidad de aprovechar los programas de análisis realizados para implementarlos de manera automática en la red, de tal manera que se actualicen los resultados con los nuevos registros de tráfico automáticamente. También se plantea el uso de estos análisis para detectar anomalías del tráfico recogido y con ello poder detectar posibles problemas de seguridad en la red.

Otra futura línea de trabajo se plantea para utilizar las herramientas de *BigData* en análisis sobre tráfico a tiempo real.

LISTADO DE ACRÓNIMOS

ACK*: Acknowledgement.*

CDN*: Content Delivery Network.*

DAG: Gráfico acíclico dirigido.

DNS*: Domain Name System.*

FQDN*: Fully Qualified Domain Name.*

FTP*: File Transfer Protocol.*

HDFS*: Hadoop Distributed File System.*

HTTP*: Hypertext Transfer Protocol.*

IANA*: Internet Assigned Numbers Authority.*

IP*: Internet Protocol.*

ISP*: Internet Service Provider*

LAN*: Local Area Network.*

NAT*: Network Address Translation.*

OSI*: Open System Interconnection.*

RDD*: Resilient Distributed Data.*

RST*: Reset.*

RTT*: Round Trip Time.*

SMTP*: Simple Mail Transfer Protocol.*

SQL*: Structured Query Language.*

SSH*: Secure Shell.*

TCP*: Transport Control Protocol.*

TFG: Trabajo de Fin de Grado

TLS*: Transport Layer Security.*

UDP*: User Datagram Protocol.*

URL*: Uniform Resource Locator.*

YARN*: Yet Another Resource Negotiator.*

WWW: *World Wide Web*

BIBLIOGRAFÍA

[1] Big Data Polito, sitio web. http://bigdata.polito.it/, (Acceso 15 de noviembre 2016).

[2] Tom White. Hadoop: the definitive guide, 4th edition. O'Reilly, 2015.

[3] Tstat, sitio web. Página principal: http://tstat.polito.it/ (Acceso 15 de noviembre 2016).

[4] Paolo Garza, Apuntes de clase: *Big Data: architectures and data analysis,* 2015. *http://dbdmg.polito.it/wordpress/teaching/big-data-architectures-and-data-analytics/* (Acceso 20 octubre 2016).

[5] Edward Capriolo, Dean Wampler, and Jason Rutherglen. *Programing Hive.* O'Reilly, 2012.

[6] Hadoop, sitio web. Página principal: http://hadoop.apache.org/ (Acceso 25 octubre 2016).

[7] Spark, sitio web. Página principal: http://spark.apache.org/ (Acceso 15 de noviembre 2016).

[8] Hive, sitio web. https://hive.apache.org/ (Acceso 10 de noviembre 2016).

[9] hdfs command, sitio web. https://hadoop.apache.org/docs/r2.7.1/hadoop-project-dist/hadoop-common/FileSystemShell.html (Acceso 15 de noviembre 2016).

[10] OBS Business School Studio Big Data, sitio web. http://www.obs-edu.com/es/noticias/estudio-obs/en-2020-mas-de-30-mil-millones-de-dispositivos-estaran-conectados-internet . (Acceso 5 de noviembre 2016).

[11] CNBC Big Data applications on a medical study, sitio web: http://www.cnbc.com/2013/09/13/big-datas-powerful-effect-on-tiny-babies.html . (Acceso 5 de Noviembre 2016).

[12] RFC 791 protocolo IP, sitio web. https://rfc-es.org/rfc/rfc0791-es.txt. (Acceso 10 de noviembre 2016).

[13] RFC 793 protocolo TCP, sitio web, https://www.rfc-es.org/rfc/rfc0793-es.txt (Acceso 10 de noviembre 2016).

[14] RFC 2616 protocolo HTTP, sitio web, https://tools.ietf.org/html/rfc2616 (Acceso 10 de noviembre de 2016).

[15] Artur Ziviani, An Overview of Internet Measurements: Fundamentals, Techniques, and Trends.

http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=27E239718FCA8DB7AA1BBCF40FCF63C6?doi=10.1.1.90.1180&rep=rep1&type=pdf (Acceso 5 de enero 2017).

[16] IANA, lista de los puertos conocidos de TCP, sitio web, http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml (Acceso 12 de enero de 2017)

ANEXOS

Anexo 1

Este anexo contiene los *scripts* utilizados en Linux.

1 Copy.sh

Este archivo recibe como parámetros el directorio donde se encuentran los archivos, el directorio final al que deben ser copiados y el nombre del archivo a copiar y realiza la copia de estos archivos manteniendo los subdirectorios en los que estaban almacenados para poder ser diferenciados.

```
#! /bin/bash
#Copy files to add the partitions
if [ $# -ne 3 ]; then
    echo "Invalid parameters"
    echo "Usage: copy.sh input_folder output_folder file_copy"
    exit -1
 fi
input_folder=$1
#input_folder=logs
output_folder=$2
#output_folder=logs/log_tcp_complete
echo "hadoop fs -ls -h $input_folder > prueba.txt"
    hadoop fs -ls -h $input_folder > prueba.txt
echo "cat prueba.txt | tr -s ' ' | cut -d ' ' -f 8  >prueba2.txt"
    cat prueba.txt | tr -s ' ' | cut -d ' ' -f 8  >prueba2.txt
echo "hadoop fs -mkdir $output_folder"
    hadoop fs -mkdir $output_folder
file_copy=$3
location_output="/user/hernandez/$output_folder"

for linea in $(cat prueba2.txt)
do
   locat_in="$linea/$file_copy"
   #-f 7 DEPENDE
   part=$(echo $linea | cut -d '/' -f 7)
   dest="$location_output/$part"
   echo "hadoop fs -mkdir $dest"
    hadoop fs -mkdir $dest
   echo "hadoop fs -cp $locat_in $dest"
    hadoop fs -cp $locat_in $dest
done
IFS=$old_IFS     # restablece el separador de campo predeterminado
rm prueba.txt
rm prueba2.txt
```

2. create_table_from_file.sh

Este script se utiliza para crear una tabla Hive a partir de un registro de tráfico de red. Como parámetros de entrada se le especifica el archivo HDFS con los datos, la base de datos de Hive a utilizar, el nombre de la tabla y su tipo.

```
#! /bin/bash
# This file create a hive table with the format of a log save in
# the hdfs directory
# It receives as parameters the name of the file, the hive
# database to use, the name of the table to create and the hive
# type (external or managed)
# It uses the file 'file_formats.txt' to get the format of each column (int,
string …)

i=1
filename="prueba2.txt"
# Check if the introduced parameters are ok.
if [ $# -ne 5 ]; then
    echo "Invalid parameters"
    echo "Usage: create_table_from_file.sh input name_file database
table_name type
    type= EXTERNAL or MANAGED"
    exit -1
 fi

table_name=$4
database=$3
name_file=$2
input=$1

# Get kind of the table
case $5 in
  external )
    type=$5
    ;;
  EXTERNAL )
    type=$5
    ;;
  MANAGED )
    type=""
    ;;
  managed )
    type=""
    ;;
  *)
    echo "Type of table invalid"
    echo "Usage: create_table_from_file input name_file database table_name
type
    type= EXTERNAL or MANAGED"
    exit -1
    ;;
 esac

#write the hive file to execute
echo "hive -e \"USE $database;" >table_create.sh
echo "CREATE $type TABLE $table_name (" >>table_create.sh
```

```
  hdfs dfs -copyToLocal $input $name_file
  #get the first line of the file and delete the initial '#'
  zcat "$name_file"| head -n 1 | cut -d '#' -f 3 >prueba2.txt
  while read line; do
      for word in $line; do
      #get name of the column
      name="$(echo "$word" | cut -d ":" -f 1)"

      #search the name of the column to find the correct format in the file
'file_formats.txt'
      # And write the hive query of the column:
      # "name format comment index"
      cat file_formats.txt | awk -v wo=$name -v ind=$i '($1 ~ wo) &&
(length($1)==length(wo))){print wo " " $2 " COMMENT \47" ind"\47," >>
"table_create.sh"}'
      i=$((i+1))
      done
  done < "$filename"
  #delete the last character ','
  sed -i '$ s/.$//' table_create.sh
  #Finish the hive query, with the partitions
  echo ")
  COMMENT '$5'
  PARTITIONED BY (day int, hour int)
  ROW FORMAT DELIMITED FIELDS TERMINATED BY ' '
  tblproperties(\\\"skip.header.line.count\\\"=\\\"1\\\")
  ;\"" >>table_create.sh
  #FOR FILE LOG_tcp_COMPLETE CHANGE:
  #tblproperties(\"skip.header.line.count\"=\"1\")
  #ROW FORMAT DELIMITED FIELDS TERMINATED BY ' '
  #FOR FILE LOG_HTTP_COMPLETE CHANGE:
  #tblproperties(\"skip.header.line.count\"=\"2\")
  #ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
  rm prueba2.txt

  #Execute the Hive query
  sh table_create.sh
```

3. Load partition

Este script se utiliza para cargar los registros de tráfico por particiones en una tabla *Hive* de tipo *managed.*

Como parámetros de entrada se introducen el directorio donde se encuentran los datos, la base de datos a utilizar en *Hive* y el nombre de la tabla en el que se cargarán.

```
  #! /bin/bash
  # This file loads partitions into a managed hive table.
  # As parameters it receives the input folder where the data is
  # stored, the hive database of the table and the name of the
  # table
  if [ $# -ne 3 ]; then
```

```
    echo "Invalid parameters"
    echo "Usage: load_partition.sh input_folder database table_name"
    exit -1
 fi
input_folder=$1
database=$2
table=$3

#save hive query in a text document to execute it
echo "USE $database;" > hive_table.txt
j=10
#get the logs by days to separate them
#2016_02_01_* first day

for i in `seq 1 29`; do
    if [ "$i" -lt "$j" ]
    #-lt lower that (<)
    then
          #get the name of file at the directory
          echo "hadoop fs -ls -d $input_folder/2016_02_0$i*/ > prueba.txt"
          hadoop fs -ls -d $input_folder/2016_02_0$i*/ > prueba.txt
    else
          echo "hadoop fs -ls -d $input_folder/2016_02_$i*/ > prueba.txt"
          hadoop fs -ls -d $input_folder/2016_02_$i*/ > prueba.txt
    fi

    #get the location

    echo "cat prueba.txt | tr -s ' ' | cut -d ' ' -f 8  >prueba2.txt"
    cat prueba.txt | tr -s ' ' | cut -d ' ' -f 8  >prueba2.txt

    a=0 #counter

    for linea in $(cat prueba2.txt)
    do
       locat="/user/hernandez/$linea"
       echo  "load  data  inpath  '$locat'  OVERWRITE  INTO  TABLE  $table
partition(day='$i', hour='$a');" >>hive_table.txt
       a=$((a + 1))
    done

    rm prueba2.txt
    rm prueba.txt

done
#Execute hive query and save the time used
{ time hive -f "hive_table.txt" ; } 2> time2.txt
```

4. Partition.sh

Este script se utiliza para cargar los registros de tráfico por particiones en una tabla *Hive* de tipo *external*.

Como parámetros de entrada se introducen el directorio donde se encuentran los datos, la base de datos a utilizar en *Hive* y el nombre de la tabla en el que se cargarán.

```
#! /bin/bash
# This file loads partitions into an external hive table.
# As parameters it receives the input folder where the data is
# stored, the hive database of the table and the name of the
# table
if [ $# -ne 3 ]; then
    echo "Invalid parameters"
    echo "Usage: partition.sh input_folder database table_name"
    exit -1
 fi
input_folder=$1
database=$2
table=$3
#get the logs by days to separate them
#2016_02_01_* first day
#save hive query in a text document to execute it
echo "USE $database;" > hive_table.txt
j=10
for i in `seq 1 29`; do
    if [ "$i" -lt "$j" ]
    #-lt lower that (<)
    Then
           #get the name of file at the directory

           echo "hadoop fs -ls -d $input_folder/2016_02_0$i*/ > prueba.txt"
           hadoop fs -ls -d $input_folder/2016_02_0$i*/ > prueba.txt
    else
           echo "hadoop fs -ls -d $input_folder/2016_02_$i*/ > prueba.txt"
           hadoop fs -ls -d $input_folder/2016_02_$i*/ > prueba.txt
    fi

#get the location

    echo "cat prueba.txt | tr -s ' ' | cut -d ' ' -f 8  >prueba2.txt"
           cat prueba.txt | tr -s ' ' | cut -d ' ' -f 8  >prueba2.txt

    a=0
    for linea in $(cat prueba2.txt)
    do
       locat="/user/hernandez/$linea"
       part=$(echo $linea | cut -d '/' -f 3 | cut -d '.' -f 1)
       # write Hive query
       echo  "alter  table  $table  add  if  not  exists  partition(day='$i',
hour='$a') location '$locat';" >>hive_table.txt
       a=$((a + 1))
    done
    rm prueba2.txt
```

```
    rm prueba.txt

done
#Execute Hive query and save the time used

{ time hive -f "hive_table.txt" ; } 2> time2.txt
```

Anexo 2

En este anexo se realiza un análisis de los protocolos más utilizados en la red.

Protocolo IPv4

Este estándar es ampliamente conocido y utilizado ya que es el estándar en uso para cualquier conexión a Internet. IPv4 proporciona los medios necesarios para la transmisión de datos en modo paquete denominados datagramas entre una dirección de origen y de destino. Su descripción completa se encuentra en la RFC 791 [12].

Este protocolo se encuentra en el nivel 3 del modelo OSI, el llamado nivel de red.

La cabecera IP se corresponde con la figura 3.1

Figura 0-1: Cabecera IP

La cabecera recoge el valor de las direcciones IP utilizadas. Una dirección IP es un número que identifica de manera lógica a un dispositivo dentro de una red. IP incluye las direcciones de origen y destino del paquete. En la cabecera también se incluye la longitud total del paquete mandado y el protocolo de la capa superior.

Este protocolo junto con TCP es el modelo en uso para las conexiones *web*.

Protocolo TCP

El protocolo TCP (*Transport Control Protocol*) es un estándar que define como establecer y mantener una conexión de red. TCP funciona sobre el protocolo IP, y juntos, son la base de Internet. El protocolo TCP se encuentra descrito en la RFC 793 [13].

Este protocolo está orientado a conexión, lo que implica que una conexión es establecida y mantenida durante el intercambio de mensajes. Una conexión TCP se establece mediante el envío del *three way handshake* que corresponde con el envío de tres mensajes entre los dos equipos a conectar. El primer equipo realiza el envío del mensaje

SYN para transmitir los parámetros de la conexión al receptor, este envía como respuesta un mensaje SYN/ACK el que incluye sus parámetros de conexión y un reconocimiento (ACK) de los parámetros recibidos. Para terminar de establecer la comunicación, el primer equipo envía un mensaje ACK para indicar que la comunicación ha sido establecida.

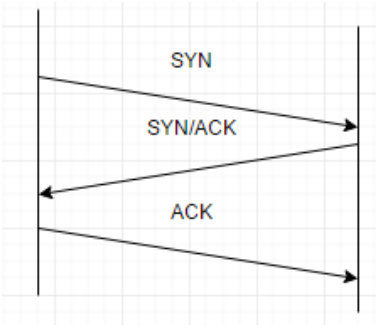


Figura 0-2 Intercambio de paquetes durante el three way handshake de TCP

Cuando un equipo desea cerrar una conexión envía el paquete FIN al cual espera recibir como respuesta un mensaje FIN/ACK. Cuando el equipo recibe este mensaje responde con un mensaje ACK que significa el fin de la conexión.

El protocolo TCP proporciona un transporte fiable de mensajes entre aplicaciones, e incluye control de flujo, control de congestión y detección y corrección de errores.

Protocolo HTTP

Hypertex Transfer Protocol es el protocolo de transferencia de texto utilizado en la *world wide web* (www). Su principal objetivo es permitir la transferencia de archivos entre un navegador (cliente) y un servidor *web*. El protocolo se encuentra definido en la RFC 2616 [14], versión 1.1.

La comunicación en este protocolo utiliza un modelo cliente servidor en la cual el navegador realiza una petición HTTP al servidor y recibe una respuesta HTTP. Este protocolo funciona sobre el modelo TCP/IP por lo que es orientado a conexión, realizada en el puerto 80 del servidor.

HTTP tiene definidos una serie de métodos de petición. El método más utilizado corresponde con el método GET, que solicita un documento al servidor. También son altamente utilizados los métodos HEAD, en el cual pide solo las cabeceras HTTP, y POST, donde se mandan datos al servidor.

Como respuesta un servidor HTTP puede incluir diversos códigos identificados mediante un número de tres cifras. Los códigos con formato 1xx incluyen respuestas

informativas; 2xx indica que la respuesta es correcta; 3xx indica redirección y 4xx y 5xx corresponden con códigos de error.

El protocolo HTTP no incorpora cifrado en sus mensajes. Para solucionar esto, se desarrolla el protocolo HTTPS, este incorpora TLS (Transport Layer Security) para proveer a la comunicación de seguridad a nivel de transporte. Estas comunicaciones se realizan sobre el puerto 443 del servidor y viajan cifradas por la red, por lo que no se puede monitorizar el valor de su campo de datos.

Protocolo DNS

El objetivo de *Domain Name System* es proveer de un mecanismo para nombrar recursos de tan manera que estos nombres puedan ser utilizados en diferentes equipos y redes.

DNS es una base de datos jerárquica y distribuida que asocia los nombres de dominio a direcciones IP que pueden ser utilizadas por los dispositivos para establecer una comunicación.

La estructura de DNS es una estructura jerarquizada en la que se encuentran en la parte superior los dominios de nivel superior o TLD (Top Level Domain), debajo de estos se encuentran los dominios de segundo orden, y en la parte baja los *host.*

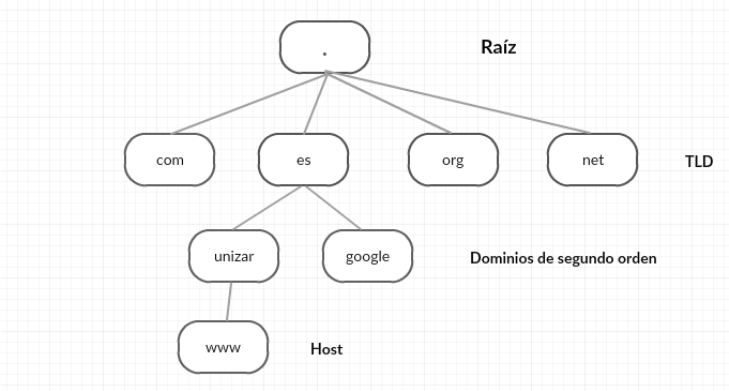


Figura 0-3 Estructura de DNS

El nombre absoluto FQDN incluye todas las etiquetas de nodo de una estructura arbórea separadas por puntos, como por ejemplo: www.unizar.es.

Desde el punto de vista del usuario, cuanto este desea conocer la resolución de nombre de un dominio, manda una petición al servidor DNS configurado en su red. Este servidor inicia la búsqueda de manera recursiva del dominio DNS si no lo tiene almacenado en caché realizando una petición a los servidores jerarquizados de dominio. Una vez obtenida la respuesta le comunica al usuario el resultado.

Los mensajes que utiliza DNS van sobre la capa de transporte y pueden ir tanto sobre UDP como TCP, aunque generalmente utilizan el protocolo UDP con conexiones al puerto 53 del servidor. Estos mensajes no incluyen ningún cifrado por lo que es posible al monitorizar una red obtener su contenido.