Universidad Autónoma de Madrid

Escuela Politécnica Superior

Doble Grado en Ingeniería Informática y Matemáticas

TRABAJO DE FIN DE GRADO

TÉCNICAS BIG DATA PARA EL ANÁLISIS DE DATOS DE TRÁFICO DE RED SOBRE HADOOP

Rubén García-Valcárcel Sen
Tutor: Dr. Iván González Martínez

Junio 2015

TÉCNICAS BIG DATA PARA EL ANÁLISIS DE DATOS DE TRÁFICO DE RED SOBRE HADOOP

Autor: Rubén García-Valcárcel Sen
Tutor: Dr. Iván González Martínez

High Performance Computing and Networking Research Group
Dpto. de Tecnología Electrónica y de las Comunicaciones
Escuela Politécnica Superior
Universidad Autónoma de Madrid

Junio 2015

Agradecimientos

Me gustaría dedicar este trabajo a todas aquellas personas que, de alguna forma u otra, han allanado este último tramo del camino.

A mi madre, Esther, y a mis abuelos, Jose Luis y Mercedes, por su generosidad sin límites, y por su apoyo incondicional durante toda mi vida.

A Nuria, por aguantar mis continuas quejas mientras escribía este documento, por animarme cada día para lograr mis objetivos, y por haberme cambiado la vida.

A mi tutor Iván, por todos los consejos tanto académicos como personales, por ser tan comprensivo, y por toda la ayuda prestada durante el desarrollo de este trabajo.

A los miembros de "El Pozo", por las intensas discusiones a la hora de comer, y por sacarme tantas sonrisas.

A Juan, con quien he compartido las dudas y preocupaciones sobre el, hasta ahora, incierto futuro a corto plazo, por su gran ayuda tanto a la hora de realizar este trabajo como al revisarlo.

A Rafael, por sus incansables ganas de ayudar, y por que sin su apoyo este trabajo habría sido muy distinto.

A Muelas, por compartir su infinita sabiduría.

A los demás miembros del grupo de investigación HPCN, ya que sin su ayuda y medios este proyecto no hubiera sido posible.

Gracias.

"Experience without theory is blind, but theory without experience is mere intellectual play."

Immanuel Kant

Resumen

La monitorización y el análisis del tráfico de red cobran gran importancia cuando se trata de optimizar los recursos de red y mejorar la experiencia de los usuarios. Las técnicas habituales utilizan un único servidor que, a pesar de ser de alto rendimiento, consta de unos recursos limitados y no es escalable para el análisis exhaustivo de grandes volúmenes de datos. Por ello, una paralelización del trabajo puede resultar útil a la hora de realizar este tipo de tareas.

El proyecto *Apache Hadoop*, junto a todas las herramientas que surgen a su alrededor, proporciona una plataforma para la computación distribuida de *Big Data* de forma escalable y fiable. El objetivo de este trabajo es la evaluación tanto del rendimiento de dicho proyecto como de las posibilidades que ofrece en el contexto de la monitorización y el análisis de redes. Con este propósito, se ha creado una herramienta basada en *Hadoop* para la captura, el almacenamiento, el procesamiento y el análisis de grandes cantidades de tráfico de red. Para probar el funcionamiento del sistema propuesto, se ha utilizado tráfico real obtenido de la red de los laboratorios docentes de la Escuela Politécnica Superior de la Universidad Autónoma de Madrid.

Se ofrece también la posibilidad de utilizar una interfaz web para definir y ejecutar consultas de forma interactiva, con las que realizar la parte del análisis. Se propone la herramienta *Apache Hive* para el desarrollo de esta parte, ya que su lenguaje de programación, basado en SQL, permite consultar de forma distribuida grandes cantidades de datos. Esto facilita la labor de los administradores de red, pues disponen de una forma sencilla de definir consultas en función de sus necesidades.

Para evaluar la capacidad del sistema creado se han realizado varios análisis sobre las diferentes fuentes de información disponibles. Estos han permitido generar estadísticas del uso de la red, detectar errores en sus componentes y realizar predicciones sobre el patrón de uso de la misma. Por último, se ha comparado el rendimiento de la herramienta propuesta con el de las soluciones actuales. Los resultados experimentales muestran que, con un pequeño clúster, se puede conseguir un tasa de procesamiento superior a 7 Gbps ($7 \cdot 10^9$ bits por segundo), mejorando el rendimiento de las herramientas más potentes disponibles en la actualidad.

Palabras clave — Big Data, computación distribuida, Hive, escalabilidad, fiabilidad, disponibilidad, análisis de datos.

Abstract

Network traffic monitoring and analysis loom large when it comes to optimizing network resources and improving the user experience. The standard techniques use a single high performance server which has limited resources and is not scalable for the thorough analysis of large data sets. Thus, a parallelization of the work can be useful when performing such tasks.

The *Apache Hadoop* project, along with all the tools that appear over it, provides a framework for the distributed computing of *Big Data* in a scalable and reliable way. The aim of this work is to evaluate both the performance of the project and the possibilities that it offers in the context of network traffic monitoring. To this end, we have created a tool based on *Hadoop* in order to capture, store, process and analyze huge amounts of network traffic. For the purpose of testing the performance of this tool, we have used real network traffic which has been obtained from the laboratories' network of the *Escuela Politécnica Superior* in the *Universidad Autónoma de Madrid*.

We also offer the possibility to use a web interface to define and execute queries interactively, in order to perform the analysis. We propose the *Apache Hive* tool for the development of this part, as its programming language, based on SQL, provides a way to query large amounts of data in a distributed manner. This facilitates the work of network administrators, since they have an easy way to define queries based on their needs by using the proposed system.

To assess the possibilities of the created system, we have conducted several analysis that use the available sources of information. These analysis have generated several statistics about the network usage and they have detected some errors on its components. Besides, we have been able to make some predictions about the use pattern of the network. Finally, we have compared the performance of the proposed tool with the current solutions. Experimental results show that, with a small cluster, it is possible to obtain a performance higher than 7 Gbps ($7 \cdot 10^9$ bits per second), improving the performance of the most powerful tools that are available nowadays.

Key words — Big Data, distributed computing, Hive, scalability, reliability, availability, data analysis.

Glosario

Big Data Conjuntos de datos tan grandes o complejos, que superan la capacidad del *software* habitual para ser capturados, gestionados y procesados en un tiempo razonable. 10, 37

Disector Programa informático capaz de leer tráfico de red y decodificar protocolos de red. 6, 10, 12, 16–18, 20, 23, 25–27, 30, 39, 40, 42–45, 49

Disponibilidad Probabilidad de que un sistema pueda ser accedido durante un periodo de tiempo determinado. 2, 18

Escalabilidad Capacidad de una aplicación para adaptarse fácilmente a una mayor necesidad de trabajo. 2, 5, 6, 9, 16, 18, 43, 44, 48, 49

Fiabilidad Probabilidad de que un sistema desarrolle una cierta función correctamente, bajo condiciones fijadas y durante un período de tiempo determinado. 2, 6, 17, 18, 43, 48

Flujo Secuencia de paquetes con características comunes que pasan por el mismo punto de la red. En este trabajo, dichas características son las direcciones IP, los puertos TCP o UDP de origen y de destino, y el protocolo de nivel de transporte. 6, 7, 11, 15, 17, 18, 23–25, 29–31, 47, 49

Gbps Unidad de transferencia, gigabits (10^9 bits) transferidos en un segundo. 6, 9, 16, 23, 39–41, 44, 47, 50

MapReduce Paradigma de programación paralela que divide un problema en subtareas (*Map*) cuyos resultados parciales agrupa (*Reduce*) para obtener la solución o las soluciones finales del problema. 6, 10, 12, 13, 17, 20, 24, 26, 29, 39, 48, 50

PCAP Formato de almacenamiento de paquetes de tráfico de red utilizado por la herramienta libpcap. 6, 7, 11, 12, 15–21, 23–27, 29, 30, 39, 41, 47, 49

Acrónimos

API Application Programming Interface. 7, 17–19, 24–26, 49

CSV Comma-Separated Values. 15, 16, 23, 26

DNS Domain Name System. 7, 13, 15, 16, 18, 23, 25–27, 29, 43, 44, 47–49

DPI Deep Packet Inspection. 24, 25

ECDF Empirical Cumulative Distribution Function. 34

FPGA Field Programmable Gate Array. 3, 5, 10

GPU Graphics Processing Unit. 3, 5, 10, 50

HDFS Hadoop Distributed File System. 6, 10, 12, 13, 15–20, 24, 26, 39–41, 47, 49

HTTP Hypertext Transfer Protocol. 6, 7, 12, 13, 15, 16, 18, 23, 25, 26, 29, 30, 32–35, 39, 40, 43–45, 47

HTTPS Hypertext Transfer Protocol Secure. 13, 15, 16, 18–20, 23, 25–27, 29, 30, 32, 33, 43, 44, 47

IP Internet Protocol. 5, 7, 15, 16, 18, 19, 24–27, 30, 31, 35, 49

RAID Redundant Array of Independent Disks. 39–41

SQL Structured Query Language. 6, 7, 12, 29, 48

TCP Transmission Control Protocol. 7, 15, 18, 19, 25, 30

UDP User Datagram Protocol. 18, 19, 25, 31

URL Uniform Resource Locator. 16, 25, 26, 35

Índice general

1. Introducción **1**
1.1. Motivación del proyecto . . . 1
1.2. Objetivos del proyecto . . . 2
1.3. Estructura del documento . . . 3

2. Estado del arte **5**
2.1. Captura y procesamiento de tráfico . . . 5
2.2. Computación distribuida . . . 6
2.3. Análisis de tráfico con Hadoop . . . 7
2.4. Conclusión . . . 8

3. Definición del proyecto **9**
3.1. Alcance . . . 9
3.2. Metodología . . . 10
3.3. Herramientas . . . 11
3.4. Red analizada . . . 13
3.5. Catálogo inicial de requisitos . . . 14
3.5.1. Subsistemas . . . 15
3.5.2. Requisitos funcionales . . . 15
3.5.3. Requisitos no funcionales . . . 16

4. Diseño **17**
4.1. Arquitectura . . . 17
4.2. Formato PCAP en Hadoop . . . 18
4.3. MapReduce y disectores de tráfico . . . 20

5. Implementación **23**
5.1. Captura de tráfico . . . 23
5.2. Formato PCAP en Hadoop . . . 24
5.3. Disectores de tráfico . . . 26
5.4. Análisis de los datos . . . 27

6. Experimentos **29**
6.1. Análisis mediante Hive . . . 29
6.2. Tiempo entre peticiones . . . 33

6.3. Predicciones 35

7. Pruebas **39**

7.1. Pruebas de rendimiento 39

7.1.1. Clúster Hadoop 40

7.1.2. Comparativa de rendimiento 44

7.2. Pruebas de verificación y validación 44

8. Conclusiones **45**

9. Trabajo futuro **47**

Bibliografía **49**

Índice de tablas

6.1. Páginas web más solicitadas vía HTTP y HTTPS 33
6.2. Matriz de confusión de la predicción de horarios 37

7.1. Características de los sistemas de pruebas 41
7.2. Rendimiento de cada sistema de pruebas 44

Índice de figuras

3.1. Arquitectura de la red analizada 14

4.1. Arquitectura del sistema propuesto 18
4.2. Diagrama de clases de la API que lee archivos PCAP de HDFS 19
4.3. Ejemplo de ejecución de un programa MapReduce sobre archivos PCAP . . 20

6.1. Series temporales del consumo de la red 30
6.2. Bytes (exterior) y flujos (interior) usados por cada protocolo y puerto . . . 31
6.3. Las 16 asignaturas que más han consumido (periodo 02/02/15 − 01/05/15) 31
6.4. Equipos encendidos cada noche del curso (periodo 02/02/15 − 01/05/15) . 32
6.5. Equipos inactivos cada semana del curso (periodo 02/02/15 − 01/05/15) . 32
6.6. ECDF del tiempo entre peticiones 33
6.7. Histogramas del tiempo entre peticiones 34
6.8. Histogramas del tiempo entre peticiones, clasificados por la URL 35

7.1. Rendimiento de las tres versiones del disector de HTTP en SAR 40
7.2. Rendimiento del *test* de lectura en *Hadoop* 41
7.3. Rendimiento del *test* de lectura en *Hadoop* con mayor granularidad 42
7.4. Rendimiento de los disectores y el *test* de lectura en el clúster de *Hadoop* . 42
7.5. Rendimiento de los disectores de HTTP en *Hadoop* y en SAR 43

1
Introducción

Este Trabajo de Fin de Grado, realizado en colaboración con el grupo de investigación *High Performance Computing and Networking* (HPCN), tiene como propósito la evaluación del proyecto *Apache Hadoop* [1] en el contexto de la monitorización y el análisis del tráfico de red. Para ello se pretende crear una herramienta basada en dicha arquitectura y probarla en un entorno real, observando las posibilidades que ofrece y el rendimiento que se puede alcanzar. En los siguientes apartados se explican la motivación y los objetivos del trabajo, así como una breve descripción de la estructura del resto del documento.

1.1. Motivación del proyecto

En las últimas décadas se han desarrollado multitud de técnicas para la monitorización y el análisis del tráfico de red. La mayoría pueden clasificarse como activas o pasivas, cada una con sus ventajas e inconvenientes. La monitorización activa genera tráfico para realizar mediciones sobre el estado de la red y obtener una gran cantidad de información útil, pero puede llegar a interferir con el tráfico legítimo de la propia red [2]. Otra desventaja de este tipo de técnicas es que son incapaces de estudiar el comportamiento de los usuarios, así como de otros eventos no previstos en la red analizada. Por otro lado, la monitorización pasiva ofrece la capacidad de obtener métricas interesantes sin perturbar el comportamiento de la red. Es por ello que este último tipo suele preferirse a la hora de lidiar con la planificación y la gestión de redes de cierto tamaño. Así, existen en la actualidad multitud de herramientas con esta filosofía [3, 4, 5, 6].

Una de las desventajas de la monitorización pasiva de redes de altas velocidades radica en la necesidad de emplear servidores de altas prestaciones, con una gran capacidad de

almacenamiento para guardar los datos de forma persistente, y posteriormente analizarlos en un tiempo razonable. Debido al crecimiento exponencial de las redes de comunicaciones, los requerimientos mínimos de estos equipos son cada vez mayores, lo que hace menos viable el uso de las herramientas mencionadas anteriormente debido a su limitada escalabilidad. Por ello, el desarrollo actual se está orientando hacia el *hardware* de alto rendimiento, que permite un análisis en paralelo de los datos [7, 8, 9].

Por otro lado, uno de los problemas que se deben tratar cuando se utiliza un sistema de alto rendimiento es su disponibilidad, que suele verse reducida como consecuencia del uso intensivo de los recursos. De igual forma, el aumento del número de componentes de un sistema incrementa la probabilidad de fallo en alguno de ellos. Adicionalmente, el almacenamiento de los datos de red resulta de vital importancia; un análisis detallado del estado de la red puede dar información acerca de los problemas en sus diferentes componentes, además de posibilitar la detección de intrusiones, ataques por denegación de servicio o futuros problemas de escalabilidad. Para lograr este objetivo, es necesario mantener en todo momento varias réplicas de la información en un sistema de ficheros distribuido.

Por todos los motivos antes mencionados, se hace necesario disponer de nuevas herramientas capaces de escalar al ritmo del crecimiento de la cantidad de información, además de cubrir las necesidades de los administradores de red en cuanto a simplicidad y funcionalidad. Asimismo, es importante la redundancia de los datos y la tolerancia frente a fallos del sistema sobre el que se ejecutan las nuevas herramientas.

1.2. Objetivos del proyecto

Como consecuencia de las necesidades mencionadas en la sección anterior, los principales objetivos planteados en este Trabajo de Fin de Grado son:

- Diseñar un sistema para la captura, el almacenamiento, el procesamiento y el análisis de grandes cantidades de tráfico de red. Este deberá incluir la tecnología ofrecida por el proyecto *Apache Hadoop* [1], que permite realizar computación distribuida de forma escalable y fiable. De cara a probar esta herramienta dentro del contexto de la monitorización de redes de comunicaciones, se pondrá en marcha un prototipo de captura y procesamiento de tráfico dentro de la red local de laboratorios de la Escuela Politécnica Superior de la Universidad Autónoma de Madrid.

- Analizar los datos disponibles tras su captura y procesamiento. Dado que el perfil de los usuarios a los que va destinado este sistema son los administradores red, se les proporcionará algunas consultas de ejemplo. Es importante que estas sean significativas y fácilmente modificables en la medida de lo posible, para simplificar el proceso de adición o modificación de las mismas. Adicionalmente, se intentará llevar a cabo un análisis algo más sofisticado, con el objetivo de aplicar técnicas de aprendizaje automático. Si se logra modelar el comportamiento de los usuarios, se

facilitaría el proceso de optimización de recursos de red, mejorando así la calidad de servicio.

- Evaluar el rendimiento del sistema propuesto. Aunque el proyecto *Apache Hadoop* no fue diseñado con el rendimiento como principal objetivo, se realizará una comparativa con el de otras soluciones. Se utilizará para ello una implementación en serie [10], y dos versiones disponibles en la literatura que realizan un trabajo similar, utilizando también la arquitectura de *Hadoop* [11, 12]. Además, se intentará superar el rendimiento de todas estas implementaciones. Sin embargo, no se considerarán plataformas como FPGAs o GPUs, cuyo *hardware* sí que está orientado al rendimiento.

1.3. Estructura del documento

En el capítulo 2 se realiza un análisis del estado del arte. Para ello se estudian las capacidades y los problemas de algunas de las herramientas existentes para la captura y el procesamiento del tráfico de red, así como el proyecto *Apache Hadoop* y las herramientas y proyectos que surgen a su alrededor.

En el capítulo 3 se definen y acotan las características del proyecto. Se describen la metodología y las herramientas utilizadas para el desarrollo del mismo, además de la red donde se instalará el sistema de captura. Por último, se precisan cada uno de los subsistemas que componen el proyecto, y se enumeran los requisitos tanto funcionales como no funcionales que este deberá cumplir.

En el capítulo 4 se describen la arquitectura y el diseño del sistema, y en el 5, el proceso de implementación de los módulos de captura y procesado de datos en *Hadoop*. En el capítulo 6 se muestran las consultas y otros tipos de análisis de los datos disponibles.

En el capítulo 7 se evalúa el rendimiento de la arquitectura propuesta y de la herramienta implementada. También se describen las pruebas realizadas para verificar el correcto funcionamiento del sistema. Por último, en el capítulo 8 se muestran las conclusiones sobre el trabajo realizado, y en el capítulo 9, las diferentes líneas de mejora y trabajo futuro.

2

Estado del arte

Como se ha mencionado en el capítulo anterior, la monitorización pasiva de redes es una tarea cuya complejidad depende directamente del tamaño de las mismas. En este capítulo se presentan las herramientas actuales más populares que permiten llevar a cabo este tipo de monitorización, así como algunas de las alternativas existentes que mitigan los problemas de escalabilidad. Se comenta el proyecto *Apache Hadoop* como solución para la computación distribuida, así como algunas de las herramientas que han surgido sobre el mismo, y las soluciones existentes para el procesamiento de tráfico de red que hacen uso de esta arquitectura.

2.1. Captura y procesamiento de tráfico

Generalmente, la monitorización de redes de altas prestaciones se lleva a cabo en servidores dedicados, cuyas características exceden en gran medida de las del *hardware* ordinario. Algunas de las herramientas más populares para la captura de tráfico son *Tcpdump* [3], *Wireshark* [4] o *CoralReef* [5]. Sin embargo, ninguna de ellas es capaz de alcanzar las altas tasas de red actuales, por lo que una vez superado el umbral que permiten procesar, comienzan a descartar paquetes. Para superar estas limitaciones surgen alternativas basadas en *hardware* específico como los dispositivos de *hardware* programable (FPGAs) [7] o las unidades de procesamiento gráfico (GPUs) [8, 9], algunas de las cuales son capaces de procesar a tasas de red muy superiores a las necesarias de hoy en día.

El protocolo *NetFlow* de *Cisco* [6], define una forma de agrupar los paquetes que comparten características como las direcciones IP y los puertos, reduciendo así la información a almacenar e introduciendo el concepto de flujo. Una herramienta basada

en este protocolo es *FlowProcess* [13], la cual es capaz de capturar los paquetes de red y reconstruir los flujos subyacentes. Su principal ventaja con respecto al resto de herramientas del mercado es que, bajo ciertas condiciones razonables, permite alcanzar tasas de red de 10 Gbps, operando sobre un *hardware* de propósito general.

Respecto al procesamiento de los paquetes capturados, el disector de tráfico web presentado en [10] es capaz de analizar archivos en formato PCAP, reconstruyendo conexiones HTTP y exportando información sobre las mismas. Además, permite realizar estas tareas eficientemente sin necesidad de disponer de un *hardware* específico para ello, pudiendo procesar hasta 4 Gbps si se utiliza una máquina con buenas prestaciones (ver sección 7.1). No obstante, este proceso se lleva a cabo en una única máquina y utilizando un único proceso, pues la paralelización de tareas como esta es compleja. Por ello, ante un futuro incremento de las necesidades de procesamiento, herramientas como esta se quedarían obsoletas.

2.2. Computación distribuida

Debido a las necesidades de escalabilidad y fiabilidad planteadas, nuevos paradigmas de computación y de almacenamiento con estas características tan deseables comenzaron a desarrollarse. Uno de los precursores en llevar estas ideas a cabo fue *Google*, que desarrolló su propio sistema distribuido de ficheros, GFS [14], y publicó un nuevo paradigma de computación, que denominó MapReduce. La combinación de estas tecnologías hace posible el procesamiento de enormes cantidades de datos distribuidos en cientos, o incluso miles de nodos.

Basándose en esta tecnología, *Apache* crea su propio proyecto de *software* libre, *Hadoop* [1], desarrollado en *Java*. Su núcleo consta de un sistema distribuido de ficheros, HDFS [15], así como de una implementación propia del paradigma MapReduce de *Google*, conociéndose en su última versión estable como *YARN* o MapReduce 2,0 [16]. Dichas herramientas permiten paralelizar el análisis de los datos de una forma transparente y eficiente. El proceso es robusto y fiable, ya que si falla cualquier nodo, un disco o la red, el sistema continúa funcionando. La principal ventaja de estos sistemas es su escalabilidad: la simplicidad de la ampliación del clúster reside en la transparencia de los procesos de distribución de tareas y de archivos entre los diferentes nodos.

Sobre el núcleo de *Hadoop* surgen multitud de herramientas, comúnmente llamadas su *ecosistema*. Entre las más conocidas se encuentran *Apache HBase* [17] para el almacenamiento estructurado de los datos en HDFS, *Apache Pig* [18] para su análisis a alto nivel, o *Apache Mahout* [19] para realizar aprendizaje automático. La herramienta *Apache Hive* [20] permite proyectar una estructura sobre los datos disponibles, es decir, visualizar los archivos como si fuesen tablas en una base de datos. Además proporciona un lenguaje propio basado en SQL, llamado *HiveQL*. Las consultas realizadas en este lenguaje son traducidas automáticamente al paradigma MapReduce. Por ello, esta herramienta es una buena opción para los analistas de datos con conocimientos de SQL que deseen probar *Hadoop*, ya que permite realizar de manera muy simple un potente análisis de una gran

cantidad de información y de forma distribuida.

Hadoop y su *ecosistema* se están extendiendo día a día, pues multitud de grandes empresas como *eBay*, *Facebook*, *LinkedIn* o *Yahoo!* los están utilizando [21]. Otras empresas ofrecen incluso plataformas enteras de computación sobre *Hadoop* [22]. Algunos de los ejemplos más relevantes son la infraestructura *Elastic Compute Cloud* (EC2) de *Amazon*, el *IBM InfoSphere BigInsights*, o la distribución *CDH* (*Cloudera's software distribution containing Apache Hadoop*).

2.3. Análisis de tráfico con Hadoop

El proyecto *Apache Hadoop* fue diseñado para la realización de un procesamiento por lotes, con aplicaciones en diversos ámbitos como la minería de datos, la indexación de páginas web o el análisis de *logs*. Por ello la lectura de ficheros de texto plano está muy simplificada en esta infraestructura. Sin embargo, este no suele ser el formato en el que se suele guardar el tráfico de red. A pesar de que *Hadoop* incorpora una forma de leer ficheros binarios, estos deben estar en un formato determinado. Así, no está preparado para interpretar los formatos estándares de almacenamiento de tráfico, como es el caso del PCAP, pues en principio no sabe cómo extraer, de forma individual, los paquetes que contienen. Por ello, el análisis de este tipo de ficheros se vuelve algo más complejo. Una forma de resolver esto es realizar una conversión al formato binario que proporciona *Hadoop*. No obstante, esta solución es ineficiente, pues requiere demasiado tiempo y espacio de almacenamiento.

Por estos motivos, algunos investigadores comenzaron a desarrollar sus propias librerías para lidiar con este problema y abrir archivos en formato PCAP con *Hadoop*. El trabajo de RIPE [11] ha sido uno de los primeros en mezclar el mundo de la computación distribuida y *Hadoop* con el mundo de las redes y su monitorización. Este consiste en una API que permite extraer la información de los paquetes, con los campos de algunos protocolos de diversos niveles OSI. No obstante, esta implementación solo permite interpretar los protocolos de nivel de aplicación DNS y HTTP, para lo cual utiliza bibliotecas de terceros. Además, la implementación no es eficiente, pues gasta demasiado tiempo y memoria en la reconstrucción de paquetes IP y flujos TCP.

Un trabajo más reciente es el que presenta Y. Lee en [12]. En él se detallan las características de un sistema de monitorización capaz de procesar, mediante una API también propia, tráfico IP, TCP, HTTP y registros de flujos en formato *NetFlow*. Su implementación permite procesar 60 Mbps por cada core disponible en el clúster, consiguiendo hasta $3,5$ veces más rendimiento que la librería de RIPE. Por último, en [23], Y. Lee presenta una heurística mediante la cual es posible fragmentar archivos PCAP en diversos bloques, de modo que *Hadoop* pueda procesar un único archivo de varios gigabytes o terabytes de forma paralela en diferentes nodos.

2.4. Conclusión

Existe una gran variedad de herramientas que permiten realizar los procesos de captura de tráfico y el procesamiento. Sin embargo, ninguna es mejor que el resto de forma absoluta, si no que cada una es ventajosa en diferentes situaciones. Existe también mucha literatura sobre la computación distribuida, y en particular sobre el proyecto *Hadoop* y todo su *ecosistema*. No obstante, el uso de esta infraestructura en el contexto de la monitorización de redes es un campo muy joven y poco explorado. Consecuentemente, se esperan encontrar problemas no resueltos al lidiar con la apertura de los archivos binarios de trazas en *Hadoop*.

3

Definición del proyecto

En este capítulo se analizan, de una forma más detallada, los objetivos del Trabajo de Fin de Grado. En primer lugar, en el alcance del proyecto se especifica y acota el trabajo que se pretende realizar. Posteriormente, se explica la metodología escogida para lograr cumplir los objetivos a tiempo, así como las herramientas que se ha planeado utilizar durante su desarrollo. Tras una breve descripción de la arquitectura de la red en la que se instalará el sistema de captura, se dividirá el proyecto en subsistemas y se enumerarán los requisitos tanto funcionales como no funcionales del mismo.

3.1. Alcance

El principal objetivo de este trabajo es la evaluación del proyecto *Apache Hadoop* en el contexto de la monitorización de redes. Para ello se debe diseñar un sistema, que cubra desde un primer proceso de captura hasta el análisis final de los datos. También se deberá crear un prototipo de dicho sistema para finalmente instalarla y probarla en un entorno real.

El sistema deberá ser escalable y tolerante frente a fallos. Adicionalmente, el procesamiento de los datos deberá alcanzar un rendimiento mínimo, dentro de las posibilidades de *Hadoop*. Este umbral se ha establecido en 4 Gbps, el rendimiento de la mejor versión serie de la que se dispone. Esta versión consiste en un programa codificado en *C* y ejecutado en una máquina de altas prestaciones (ver sección 7.1). Además, dado que para realizar el procesamiento se cuenta con un clúster con un gran número de núcleos, una comparación basada únicamente en el rendimiento global no sería justa. Por ello, se ha determinado también un umbral mínimo de rendimiento por cada core disponible que

se debe superar. Este consiste en el obtenido por Y. Lee en su trabajo [12], que es igual a 60 Mbps por cada core utilizado. No obstante, no se pretende superar el rendimiento que se podría llegar a obtener empleando plataformas como FPGAs o GPUs, cuyo *hardware* está orientado a optimizar dicha característica.

Por otro lado, tampoco se pretende que la herramienta sea capaz de analizar el tráfico en tiempo real, pues *Hadoop* está diseñado para realizar un procesamiento por lotes. Por ello, el procesamiento se llevará a cabo cuando se disponga de una cantidad suficientemente grande para que se pueda llamar Big Data, y sea más viable utilizar este tipo de arquitecturas frente al resto.

En cuanto al análisis, se van a estudiar las posibilidades que ofrece esta arquitectura y en particular, la herramienta *Apache Hive*. Con esto se espera cubrir diferentes ejemplos, de forma que un administrador de red pueda ser capaz de, a partir de ellos, modificarlos o crear otros nuevos con facilidad según sus necesidades. Por tanto, el objetivo del trabajo no es realizar un análisis exhaustivo de los datos disponibles.

3.2. Metodología

Con el objetivo de evaluar la complejidad del proyecto y encontrar herramientas que faciliten el proceso de desarrollo, se ha destinado un primer periodo de estudio del problema y del estado del arte. Tras esto se han podido definir y estimar, con mejor criterio, los requisitos, la planificación y la metodología que se deberá seguir para conseguir los objetivos en el limitado tiempo disponible.

En el desarrollo del proyecto se pueden diferenciar claramente tres fases:

- Implementación del módulo de captura de tráfico. Se debe comenzar a capturar cuanto antes para que, llegado el momento de probar las siguientes fases, se disponga de una cantidad suficiente de tráfico de red para aprovechar completamente las características de *Hadoop*. Por ello, la prioridad de esta fase es máxima, y se partirá de la herramienta *FlowProcess* [13], sobre el que se implementarán los cambios necesarios para adaptarlo al sistema completo. La complejidad de esta fase no se prevee elevada.

- Desarrollo de disectores de tráfico que procesen los datos de red almacenados en HDFS. Este es un proceso complejo, ya que se debe entender en primer lugar el funcionamiento de la lectura de ficheros en HDFS, así como el de los disectores de tráfico y del paradigma de programación MapReduce. A pesar de que se dispone de trabajo previo que facilita este proceso de lectura, es muy posible que surjan problemas o imprevistos, pues este trabajo es muy reciente.

- Evaluación de la capacidad de análisis del sistema. Esta es la fase menos acotada, en la que se profundizará en mayor o menor medida según el tiempo disponible. Debido a la existencia de ejemplos *online* de consultas utilizando la herramienta

Apache Hive, la creación de las mismas no debería suponer una gran dificultad. No obstante, este proceso es el menos mecánico y más creativo, dado que las consultas a implementar tienen una fuerte dependencia de los datos disponibles, así como de los resultados de consultas previas. Por todos estos motivos, es difícil estimar el tiempo que puede llevar realizar este análisis.

Con esta clara división, se ha elegido un modelo incremental iterativo para el proceso de desarrollo del proyecto. Este constará de tres incrementos, que se corresponderán con cada una de las fases indicadas anteriormente. La elección de este modelo permitirá disponer rápidamente de un primer prototipo con el que comenzar a capturar datos de red, que luego podrán ser procesados y analizados, respectivamente, en el segundo y tercer incremento.

Durante el desarrollo de todo el proyecto se llevarán a cabo reuniones semanales con el tutor para evaluar el estado del proyecto y planificar las tareas a realizar y resultados a obtener durante la siguiente semana.

3.3. Herramientas

Para facilitar el proceso de implementación del proyecto, se utilizarán las siguientes herramientas:

FlowProcess

Como se ha explicado en la sección 2.1, existe una gran variedad de herramientas para la captura de tráfico, cada una con sus ventajas y sus inconvenientes. *FlowProcess* permite reconstruir flujos a partir de los paquetes capturados de una interfaz de red [13], guardando dicha información como texto en claro. Es capaz además de trabajar a altas tasas de red, sin necesidad de un *hardware* específico para ello. Esto nos permite, por tanto, instalar el programa en cualquier máquina disponible, abaratando así el coste de este proceso. Lo único que se necesita para su instalación es un sistema operativo basado en *GNU/Linux*, así como disponer de la librería *libpcap* instalada.

Otra virtud de esta herramienta es que se dispone del código fuente, por lo que será posible modificarlo con el fin de adaptarlo a las posibles necesidades del proyecto.

Wireshark

Esta herramienta es capaz de capturar los paquetes de red y mostrar información sobre dicho tráfico. Permite además interpretar los archivos en formato PCAP, realizar un filtrado o una búsqueda de paquetes con propiedades determinadas y detectar aquellos que están truncados o mal formados. Como es una herramienta muy extendida y probada,

se utilizará para comprobar que se están realizando correctamente los procesos de captura y de disección de tráfico.

Disector de tráfico HTTP

Se dispone de un disector de tráfico HTTP codificado en C, probado y optimizado [10]. Este programa es capaz de, a partir de archivos en formato PCAP, identificar las conexiones HTTP y guardarlas en un fichero de texto plano. Dado que en el segundo subsistema se deberá implementar una funcionalidad como esta en *Hadoop*, se va a partir de esta versión en serie, que se adaptará al paradigma de programación MapReduce. Por ello, este programa permitirá realizar tanto pruebas de caja negra como una comparativa del rendimiento final obtenido.

Apache Hive

Apache Hive es una herramienta del *ecosistema* de *Hadoop* que permite proyectar una estructura sobre los datos disponibles, es decir, visualizar ficheros en cualquier formato como si fuesen tablas de una base de datos. Posibilita además la creación y ejecución de consultas sobre dichas tablas mediante *HiveQL*, un lenguaje propio basado en SQL. Estas consultas son traducidas automáticamente al paradigma MapReduce, lo que permite realizar un análisis distribuido de grandes cantidades de información de forma muy simplificada. Por tanto, esta herramienta será fundamental en el tercer incremento, el análisis de los datos.

CDH (*Cloudera's software distribution containing Apache Hadoop*)

Como se ha mencionado en la sección 2.2, la empresa *Cloudera* es una de las muchas que ofrecen plataformas de computación sobre *Hadoop*. Su distribución de código libre, CDH, proporciona los elementos del núcleo de *Hadoop*, además de una gran cantidad de herramientas sobre ella, de los cuales se esperan utilizar los siguientes:

- **Cloudera Manager**. Facilita en gran medida la instalación de *Hadoop* en un clúster, pues automatiza el proceso de configuración de los equipos. Tan solo se debe proporcionar acceso *root* mediante *SSH* a cada ordenador. La herramienta ofrece además una una interfaz web que permite, entre otras cosas, examinar el estado de cada máquina, añadir o quitar nodos al clúster, y gestionar los parámetros de configuración de *Hadoop*.
- **Hue**. Proporciona una interfaz web con la que analizar la información guardada en HDFS. Permite navegar por dicho sistema de ficheros, y se integra perfectamente con *Hive*: se pueden definir, guardar y ejecutar consultas de forma interactiva, así como almacenar los resultados en otras tablas, o crear gráficas con los mismos.

- **Máquina virtual**. Si no se quiere instalar todo lo necesario para desarrollar programas MapReduce, o probar *Hadoop*, *Cloudera* proporciona una máquina virtual con todos sus servicios instalados. Incluye también, por defecto, el entorno de desarrollo integrado *Eclipse*, con el que se desarrollar y ejecutar programas MapReduce.

Otras

Las siguientes herramientas se emplearán de forma marginal durante el proyecto:

- **Bash**. Este lenguaje de *scripting* se usará con el fin de automatizar los procesos de copiado de los archivos en HDFS, la ejecución de tareas MapReduce y de consultas en *Hive*. Se ha elegido por su simplicidad, así como por la experiencia previa en el uso de dicho lenguaje.

- **Matlab**. Esta herramienta proporciona una forma fácil e interactiva de representar datos [24]. Dado que permite crear scripts para generar gráficas, se utilizará durante el tercer incremento.

- **Python**. Este lenguaje de programación permite crear rápidamente complejos análisis de los datos [25]. Sin embargo, al ser interpretado, solo se podrá utilizar sobre pequeñas cantidades de datos.

- **Weka**. Esta herramienta permite realizar aprendizaje automático de una forma muy intuitiva gracias a su interfaz gráfica [26]. Por ello se planteará su uso en el tercer incremento.

3.4. Red analizada

El sistema resultante de este trabajo se va a instalar y a probar en la red de área local de los laboratorios docentes de la Escuela Politécnica Superior de la Universidad Autónoma de Madrid. Su topología se muestra en la Figura 3.1: consta de más de 700 ordenadores, que generan una cantidad de tráfico que oscila entre 150 y 260 GB cada día lectivo. Por motivos de seguridad, un cortafuegos bloquea la mayor parte del tráfico diferente de DNS, HTTP y HTTPS, por lo que el procesamiento del tráfico se enfocará únicamente en estos tres protocolos.

Por otro lado, un servidor de DNS interno gestiona todas las peticiones de las máquinas de la subred. Debido a esto, la sonda de captura es incapaz de escuchar las peticiones DNS de los diferentes ordenadores, pudiendo tan solo capturar eventualmente las peticiones del propio servidor. Por tanto no se conocen, a priori, todas las peticiones DNS realizadas, ni las direcciones de los equipos que las solicitan.

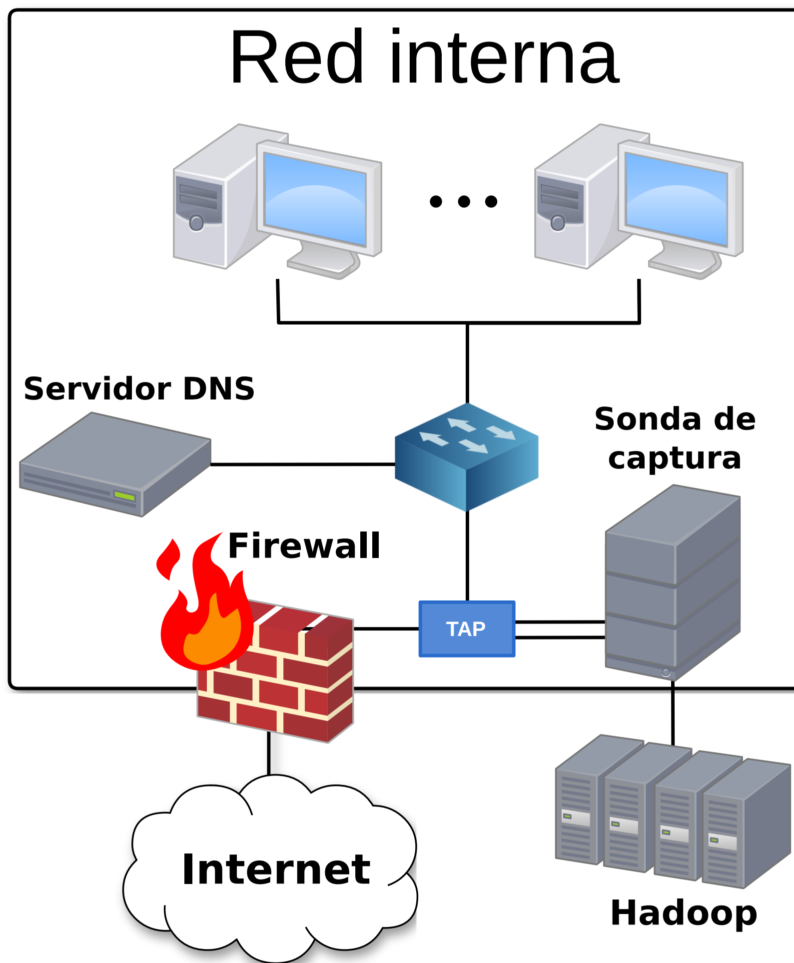


Figura 3.1: Arquitectura de la red analizada

Por último, la sonda de captura no dispone de *hardware* de altas prestaciones: consta de un procesador Intel Pentium D @ 3 GHz, así como de tan solo 2 GB de memoria RAM. Por ello, el programa de captura deberá adaptarse a estas limitaciones.

3.5. Catálogo inicial de requisitos

En esta sección se enumeran los requisitos, tanto funcionales como no funcionales, que deberá cumplir el sistema final. Con los primeros se definen las acciones fundamentales que deben tener lugar en la ejecución del *software*; los segundos, por el contrario, especifican las cualidades que el sistema debe tener, sin relación directa con su comportamiento funcional. Para realizar las pruebas de validación al finalizar cada incremento, los requisitos funcionales se han clasificado por subsistemas. A continuación se describe cada uno de ellos.

3.5.1. Subsistemas

Como ya se ha adelantado en la sección 3.2, el sistema se dividirá en tres subsistemas, correspondiendo con los tres incrementos del modelo de desarrollo planteado:

- **Subsistema de captura**. Consta de los procesos de captura de tráfico de red, de reconstrucción de los flujos subyacentes y del guardado de toda esta información en HDFS.
- **Subsistema de procesamiento**. Su propósito es la disección del tráfico capturado en formato PCAP, para obtener las conexiones HTTP y HTTPS, así como las preguntas y respuestas de DNS.
- **Subsistema de análisis**. El objetivo de esta fase es la extracción de información a partir de los datos disponibles. Para ello se crearán consultas en el lenguaje *HiveQL* propio de la herramienta *Apache Hive*. También se probarán otras tecnologías como *Matlab* para la creación de gráficas, *Python* para desarrollar de forma rápida complejos análisis o *Weka* para realizar aprendizaje automático y complejos análisis sobre una cantidad reducida de datos. Dado que esta fase es experimental, no cuenta con ningún requisito funcional.

3.5.2. Requisitos funcionales

Subsistema de captura

RF. 1 *La sonda será capaz de capturar los paquetes de una interfaz de red dada y guardarlos en formato PCAP.*

RF. 2 *La sonda reconstruirá los flujos a partir de los paquetes capturados.*

RF. 3 *La sonda deberá guardar en formato CSV al menos los siguientes datos de los flujos:*

- *Direcciones MAC de origen y destino.*
- *Direcciones IP de origen y destino.*
- *Puertos de origen y destino.*
- *Número total de paquetes y de bytes que componen el flujo.*
- *Tiempos en los que se han capturado el primer y el último paquete del flujo.*
- *Número de banderas FIN, SYN, RST, PSH, ACK, URG, CRW y ECE del protocolo TCP observadas en el flujo.*

RF. 4 *La sonda deberá mover los datos al sistema de ficheros HDFS.*

Subsistema de procesamiento

RF. 5 *La aplicación será capaz de abrir archivos en formato PCAP guardados en HDFS.*

RF. 6 *La aplicación será capaz de reconstruir las conexiones HTTP, guardando en HDFS y en formato CSV, al menos los siguientes datos de cada conexión:*

- *Direcciones IP de origen y destino.*
- *Puertos de origen y destino.*
- *Tiempos de la petición y la respuesta.*
- *URL de la petición.*
- *Código y descripción de la respuesta.*

RF. 7 *La aplicación será capaz de extraer las peticiones HTTPS, guardando en HDFS y en formato CSV, al menos las direcciones IP de origen y destino, el timestamp de cada la petición y el nombre del servidor contra el que se va a conectar.*

RF. 8 *La aplicación será capaz de diseccionar las peticiones DNS, guardando en HDFS y en formato CSV, las direcciones IP y sus nombres resueltos.*

3.5.3. Requisitos no funcionales

RNF. 1 *El proceso de captura deberá soportar una tasa de red de* 1 *Gbps sin pérdidas de paquetes.*

RNF. 2 *El proceso de copia de los datos a HDFS no deberá interferir con la red que se está monitorizando.*

RNF. 3 *El proceso de captura deberá estar preparado para su instalación en una sonda sin hardware de altas prestaciones: dispondrá de un procesador Intel Pentium D @* 3 *GHz y* 2 *GB de memoria RAM.*

RNF. 4 *Los disectores de tráfico deberán ser capaces de procesar al menos a una tasa de* 4 *Gbps.*

RNF. 5 *Los disectores de tráfico deberán ser capaces de procesar al menos a una tasa de* 60 *Mbps por cada core utilizado.*

RNF. 6 *El sistema de disección y análisis de los datos deberá ser escalable.*

RNF. 7 *El sistema de disección y análisis de los datos deberá ser tolerante frente a fallos.*

RNF. 8 *El sistema dispondrá de una interfaz web para la utilización de la herramienta Apache Hive.*

RNF. 9 *La interfaz web de gestión de consultas deberá ser interactiva e intuitiva.*

4

Diseño

El objetivo de este capítulo es describir la arquitectura y el diseño del sistema propuesto. De los tres incrementos en los que se ha dividido el desarrollo, el primero consistirá únicamente en implementar pequeños cambios en una herramienta para adaptarla al problema en cuestión, mientras que el tercero será muy experimental, sin requisitos funcionales. Por tanto, solo será necesaria una fase de diseño para el segundo incremento, el desarrollo de los disectores de tráfico. Este se ha dividido en dos partes: la API que permite la lectura de los archivos en formato PCAP de HDFS, y los programas MapReduce que procesan los paquetes leídos usando la API anterior.

4.1. Arquitectura

La arquitectura del sistema a desarrollar tiene una fuerte relación con algunos de los requisitos no funcionales definidos en la sección 3.5.3. En la Figura 4.1 se puede observar una clara división entre la captura y el resto de fases del proyecto. En primer lugar, el programa de captura de tráfico se instalará en una sonda interna de la subred analizada (ver sección 3.4). Se generarán tanto archivos de trazas en formato PCAP como ficheros de texto con los flujos reconstruidos a partir del tráfico anterior. Para maximizar la fiabilidad del sistema, diariamente se copiará toda esta información a un clúster con *Hadoop* instalado. Esto reducirá la cantidad de información perdida en caso de haber algún problema con la sonda. Dado que para efectuar esta copia se utilizará la propia red monitorizada, el proceso se realizará por las noches para no interferir con el tráfico legítimo de la red, lo cual es una característica imprescindible de las sondas de monitorización pasiva.

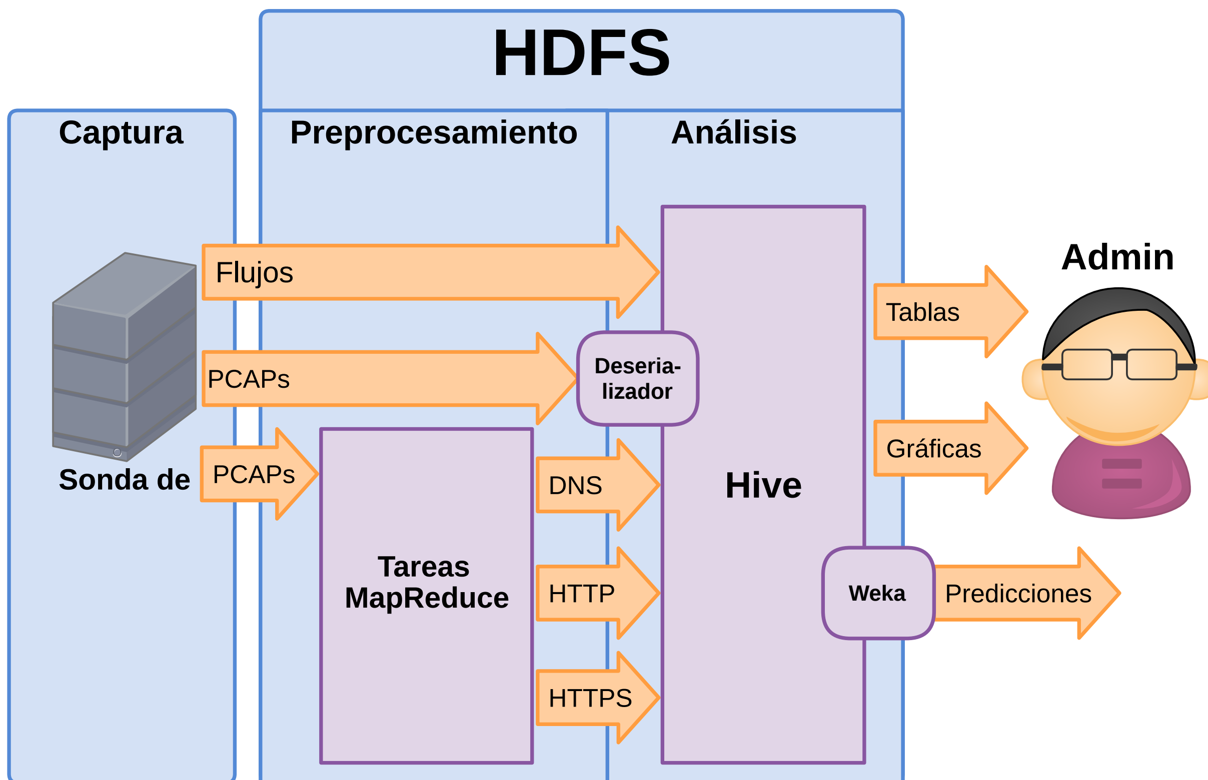


Figura 4.1: Arquitectura del sistema propuesto

Las siguientes fases, de procesamiento y análisis de los datos, se llevarán a cabo en el clúster donde se ha instalado *Hadoop*. Esto dota al sistema de la escalabilidad y la tolerancia frente a fallos requerida. Además, todos los ficheros estarán replicados y distribuidos en el clúster, con lo que se aumenta la la disponibilidad de los mismos y la fiabilidad del sistema. La entrada de la fase de procesamiento serán los archivos en formato PCAP, los cuales serán procesados por los disectores, que generarán ficheros de texto con registros de DNS, HTTP y HTTPS. Estos datos volverán a guardarse en el sistema de ficheros HDFS. Por último, la entrada de la tercera fase, el análisis, serán todos los archivos generados hasta ahora: flujos, PCAP y registros de HTTP, HTTPS y DNS. Su salida la conformarán tanto tablas como gráficas, e incluso predicciones sobre el patrón de uso de la red.

4.2. Formato PCAP en Hadoop

Como ya se ha comentado anteriormente, la apertura de archivos en formato PCAP es una tarea no trivial en *Hadoop*, ya que este proyecto no proporciona ningún método nativo para hacerlo. Para lidiar con esto, se ha partido de una API de código libre proporcionada por RIPE [11], que posibilita la decodificación de los paquetes almacenados en ficheros PCAP, extrayendo los protocolos Ethernet, IP, TCP, UDP, DNS y HTTP. Se ha elegido esta librería debido a que son los únicos que han liberado su código hasta el

momento: la implementación de Y. Lee [12], a pesar de ser más eficiente, no está disponible públicamente.

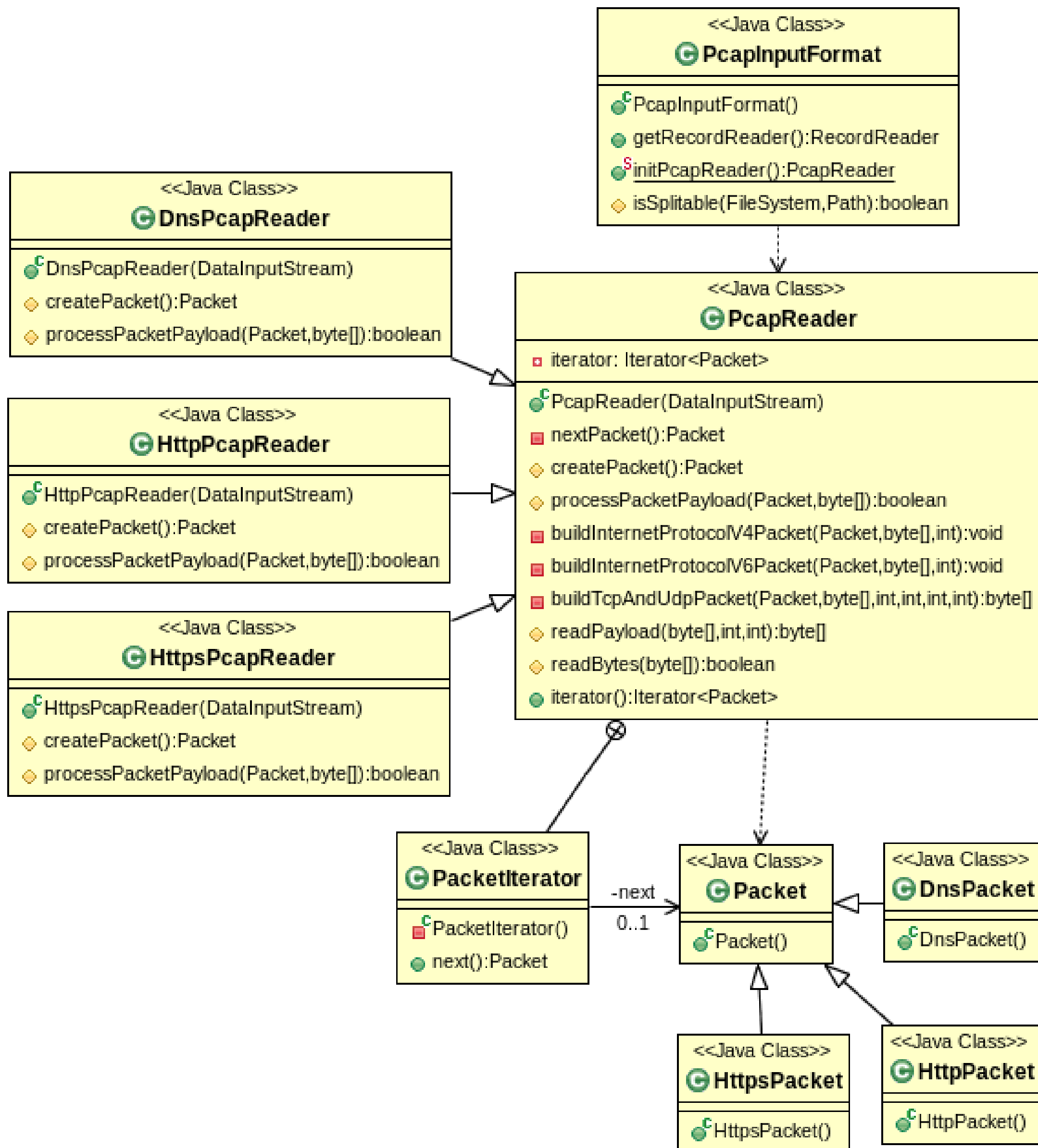


Figura 4.2: Diagrama de clases de la API que lee archivos PCAP de HDFS

Dado que una gran parte del tráfico web de hoy en día se encuentra protegido por HTTPS, se ha ampliado el proyecto de RIPE para posibilitar la extracción de dicho protocolo. La Figura 4.2 muestra un diagrama de clases de la API tras la ampliación mencionada. La clase principal es `PcapInputFormat`, encargada de definir la forma de leer los archivos de entrada: inicializa un lector de paquetes, `PcapReader`, que construye los paquetes y los devuelve de uno por uno, bajo demanda, gracias a un iterador. Este lector extrae únicamente los campos de los protocolos Ethernet, IP, TCP y UDP. Para los protocolos de nivel de aplicación, se crean subclases del lector principal, `DnsPcapReader`, `HttpPcapReader` y `HttpsPcapReader`. La última se ha añadido para posibilitar la

extracción del protocolo HTTPS. Por último, la información decodificada se guarda en las clases `DnsPacket`, `HttpPacket` o `HttpsPacket`, dependiendo del protocolo de nivel de aplicación. De nuevo, el último tipo de paquete se ha incorporado para la extracción del tráfico HTTPS.

4.3. MapReduce y disectores de tráfico

El paradigma MapReduce consiste en la división de un problema en subtareas (*Map*), ejecutadas concurrentemente para obtener unas soluciones parciales del problema, que finalmente son agrupadas (*Reduce*) para obtener las soluciones finales del problema. Casualmente, la funcionalidad de un disector se adapta perfectamente a este modelo de computación distribuida, como se explicará a continuación. En la Figura 4.3 se muestra un esquema simplificado de ejemplo de las fases de ejecución de los disectores realizados.

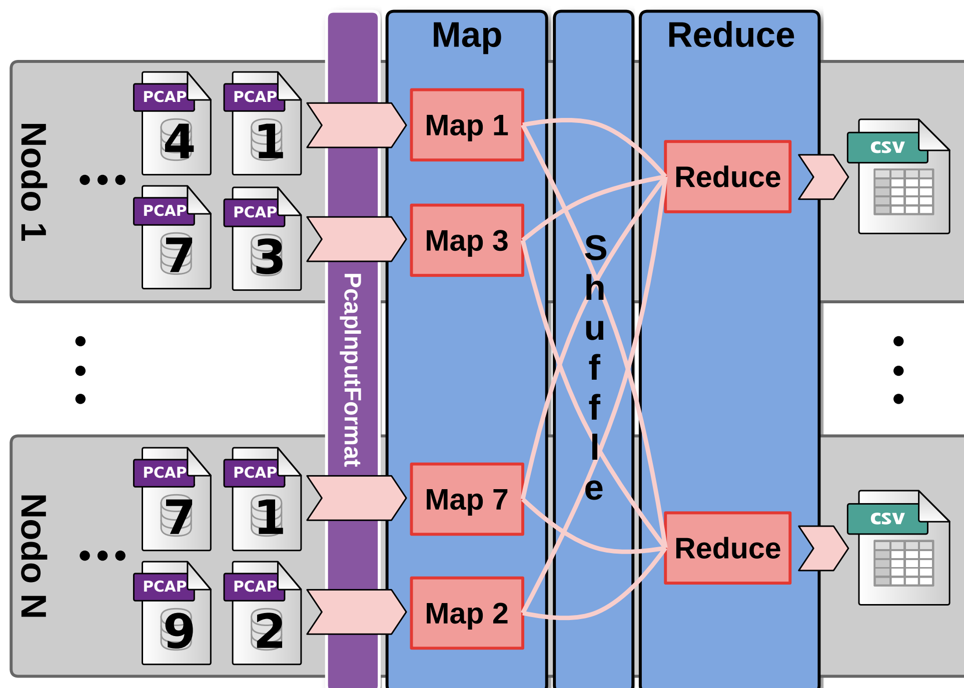


Figura 4.3: Ejemplo de ejecución de un programa MapReduce sobre archivos PCAP

En primer lugar, el planificador de *Hadoop* asigna a cada nodo los archivos que debe procesar, de forma que se lanza una tarea *Map* por archivo. Se puede apreciar que los archivos PCAP están replicados en HDFS. En ejemplo de la Figura 4.3, los ficheros 1 y 7 se encuentran simultáneamente en los nodos 1 y N. Teniendo esto en cuenta, el planificador asigna las tareas *Map* procurando mantener la localidad de los datos, es decir, de forma que cada máquina procese los archivos que almacena localmente. De no ser posible una

asignación local de las tareas, las máquinas tendrán que leer los ficheros de forma remota, utilizando la red para ello, y reduciéndose consecuentemente el rendimiento final. En el ejemplo mostrado, se asignan en primer lugar las tareas 1 y 3 al nodo 1, por lo que el nodo N no puede procesar el archivo 1, y se le asignan así las tareas 2 y 7.

Las tareas *Map*, gracias a la clase `PcapInputFormat` mencionada en la sección 4.2, disponen de un iterador que les permite recorrer los paquetes del archivo PCAP que están leyendo. Para cada paquete procesado, las tareas *Map* generan una solución intermedia compuesta por un par *(clave, valor)*. La fase *Shuffle* se encarga de agrupar todos los pares que comparten la misma clave, y enviar cada grupo a un mismo nodo para poder iniciar la siguiente etapa. Por último, en la fase *Reduce*, cada grupo de valores con la misma clave es procesado en paralelo para generar una o varias líneas de la salida final.

5

Implementación

En este capítulo se describe el proceso de implementación del proyecto, que consta de las modificaciones realizadas a la herramienta *FlowProcess* [13] y a la librería de RIPE [11], además de la implementación de los tres disectores de tráfico HTTP, DNS y HTTPS.

5.1. Captura de tráfico

Debido a la urgencia de la finalización de este primer incremento, para poder comenzar a capturar y almacenar el tráfico lo antes posible se ha utilizado la aplicación externa *FlowProcess* [13]. Esta cumple con la mayor parte de los requisitos de esta parte del proyecto: reconstruye flujos unidireccionales a partir del tráfico capturado y exporta su información en formato CSV. Además, permite realizar este proceso sin pérdidas en una máquina convencional, a tasas muy superiores de la requerida (1 Gbps), cumpliéndose dos de los requisitos no funcionales detallados en la sección 3.5.3. No obstante, se han realizado varias modificaciones a este programa para adaptarlo al problema en cuestión:

- Se deben almacenar los paquetes capturados, pues *FlowProcess* guarda únicamente los flujos reconstruidos. El formato de almacenamiento será PCAP, ya que al ser estándar podrá analizarse posteriormente con otras aplicaciones, o comprobar mediante la herramienta *Wireshark* [4] que se está guardando correctamente. Otra opción planteada es guardarlo en un formato binario propio de *Hadoop*, `SequenceInputFormat`. Esto facilitaría significativamente la apertura de estos archivos a la hora de procesarlos. Sin embargo, dado que este no es un formato estándar, las pruebas de esta fase se complicarían bastante.

- Almacenar toda la carga útil de los flujos supone un uso de disco excesivo, y necesita una velocidad de escritura elevada para que no suponga un cuello de botella. Para evitar este tipo de problemas, los paquetes serán truncados, guardándose únicamente los primeros 300 bytes. Se ha considerado que esta cifra mantiene un compromiso entre almacenamiento y capacidad de análisis, ya que en la mayoría de casos los 256 primeros bytes de la carga útil de un flujo son suficientes para clasificarlo mediante DPI [27].
- La división de archivos PCAP no es una tarea sencilla ni liviana [23], ya que los paquetes que los forman son de longitud variable, y no hay ningún símbolo que separe los diferentes paquetes, como es el caso del carácter de salto de línea en los ficheros de texto. Dado que tanto *Hadoop* cómo el paradigma MapReduce se fundamentan en el desplazamiento de la computación a los datos, es imprescindible disponer de la información repartida entre los múltiples nodos. Por tanto, una división de los archivos en esta primera fase de captura es la solución más simple y con menor coste computacional. Adicionalmente, en HDFS, los ficheros se guardan particionados en bloques, cuyo tamaño recomendado suele ser en torno a 100 MB [28]. Por ello, el valor por defecto del tamaño de estos bloques suele ser 64 o 128 MB. Un tamaño mucho menor hace que los tiempos de búsqueda de archivos y de planificación de tareas se disparen. Por otro lado, un tamaño mucho mayor puede hacer que la asignación de tareas locales a los datos sea muy costoso, o imposible. Esto es debido al incremento de la probabilidad de que el conjunto de archivos sobre los que se va a trabajar no se encuentre distribuido de forma uniforme entre los diferentes nodos de cómputo. Si esto ocurriese, algunos nodos podrían verse obligados a solicitar archivos a otros nodos, reduciéndose consecuentemente el rendimiento. Sin embargo, dicho valor por defecto está pensado para clústeres de cientos de nodos. En este trabajo, al contar con solo cinco nodos de cómputo y varios TB de información a procesar, se ha estimado que hasta un tamaño de bloque de 1 GB la localidad de los datos no supondrá un problema. Por ello, la sonda generará archivos de este tamaño y no será necesaria una división en bloques de los mismos.
- La transferencia de los datos desde la sonda hasta el clúster de *Hadoop* genera paquetes de red, que se ven forzados a pasar por la propia subred monitorizada y por tanto son capturados. Esta retroalimentación se puede evitar modificando la herramienta para que descarte todos aquellos paquetes que contengan la dirección IP de la propia sonda.

5.2. Formato PCAP en Hadoop

Como se ha comentado el la sección 4.2, para la lectura de los archivos en formato PCAP almacenados en HDFS se ha utilizado la API proporcionada por RIPE [11]. No obstante, esta implementación tiene algunos problemas que se tratarán en este capítulo: en primer lugar el rendimiento es bastante mejorable, ya que la librería propuesta por Y. Lee en su trabajo [12] llega a triplicarlo. En segundo lugar, para la extracción de los protocolos HTTP y DNS utilizan librerías externas, las cuales descartan los paquetes truncados. Dado

que la mayor parte del tráfico disponible está truncado, se deberá implementar desde cero estas dos funcionalidades. Por último, también se desarrollará una clase que decodifique el protocolo HTTPS, debido a su abundancia en el tráfico de hoy en día.

Se ha comenzado evaluando el rendimiento de esta API, con el objetivo de detectar las principales causas de su ineficiencia. Para ello se ha creado un programa muy simple, cuya funcionalidad es únicamente la de leer ficheros PCAP utilizando la librería. Se ha observado con ello que al incrementar el tamaño de la entrada, el tiempo de ejecución del programa no aumenta linealmente, si no mucho más rápido. Además, a partir de cierto tamaño, la máquina virtual de *Java* se queda sin memoria. La causa de todo esto es el reensamblado de los paquetes al lidiar con la fragmentación IP y la reconstrucción de flujos TCP: para ello los paquetes son almacenados en tablas *hash*, que solo se vacía al encontrar todos y cada uno de los fragmentos. Dado que no siempre se encuentran todos ellos, la tabla acaba ocupando demasiada memoria, y el rendimiento se ve penalizado gravemente. Sin embargo, como el tráfico disponible se encuentra truncado y solo son interesantes los primeros bytes de cada flujo, se podrá prescindir de este reensamblado.

Otra causa de la pérdida de rendimiento es la forma de almacenar los campos extraídos de los paquetes: en lugar de utilizar una estructura para hacerlo, se guardan en un array asociativo, cuya clave es el nombre del campo. Esto permite adaptar dinámicamente la información almacenada en función de cada paquete, lo que puede ser muy beneficioso si no se conoce a priori, o si solo se guardaran unos pocos campos de entre una gran variedad posible. No obstante, en este trabajo no se cumplen ninguna de las condiciones anteriores, por lo que se ha optado por cambiar esta forma de almacenamiento por estructuras, mucho más rápidas y eficientes que las tablas *hash*.

Una ventaja de la implementación de RIPE es la posibilidad indicar al principio, mediante un parámetro de configuración, el protocolo de nivel de aplicación que se quiere decodificar. Esto permite añadir nuevas clases que amplíen el repertorio de protocolos que pueden ser extraídos sin modificar el resto de código de la API. Sin embargo, los paquetes que no contienen el protocolo especificado también son devueltos, aunque sin los campos del protocolo especificado rellenos. Esto se ha modificado para que solo se devuelvan los paquetes del protocolo determinado en un principio, ya que facilitará posteriormente la tarea de los disectores.

Una vez se han implementado estos cambios en la librería de RIPE, se han codificado las clases `HttpPcapReader`, `DnsPcapReader` y `HttpsPcapReader`. Estas reciben la carga útil del protocolo de nivel de transporte (TCP o UDP), e identifican el protocolo de nivel de aplicación (HTTP, HTTPS o DNS) utilizando DPI y simples expresiones regulares.

La clase `HttpPcapReader` comprueba que la cabecera HTTP es válida según las especificaciones de este protocolo [29, 30] y extrae, para cada petición, su tipo (*GET*, *PUT*, *POST*, etc.), la versión del protocolo y la URL, compuesta por el campo *Host* (nombre del servidor) y el recurso. En el caso de las respuestas, extrae el código y la descripción de cada una. Al encontrarse los paquetes truncados, es posible que ciertas partes de la cabecera no estén presentes o que otras se encuentren incompletas. Por ejemplo, las peticiones con un recurso demasiado largo no contarán con el campo *Host*, que siempre aparece después del recurso. Esto se podrá arreglar utilizando una base de datos de peticiones DNS con

sus respuestas, que se obtendrá gracias al disector de tráfico DNS.

La clase `DnsPcapReader` obtiene todas las respuestas a cada consulta de tipo *A* (dirección de un host) y de clase *IN* (Internet), según las especificaciones del protocolo DNS [31]. Aunque el registro *CNAME* puede ser útil para la resolución final de las direcciones, se plantea como trabajo futuro dar soporte a este tipo de respuestas. Todos los demás tipos de respuestas son ignoradas, así como todas las consultas, puesto que lo único que nos interesa es la resolución de las direcciones IP.

Por último, aunque los datos del protocolo HTTPS se encuentran cifrados, se puede examinar el protocolo TLS para obtener metadatos de la conexión HTTP cifrada. La clase `HttpsPcapReader` obtiene el nombre del servidor contra el que se está conectando una máquina, examinando los mensajes de tipo *Hello* de la etapa de negociación del protocolo TLS, según sus especificaciones [32].

5.3. Disectores de tráfico

Una vez se dispone de la API que lee y decodifica el tráfico guardado en HDFS, se pueden crear programas MapReduce que reciban esta información estructurada. Ejemplos de ello son los disectores de tráfico, que procesarán cada uno de los paquetes de forma distribuida para generar una salida, como se ha explicado en la sección 4.3.

La finalidad del disector de tráfico HTTP es emparejar las peticiones y las respuestas de este protocolo, creando conexiones HTTP. Su salida es un fichero CSV con la siguiente información de cada conexión: las direcciones IP, los puertos, los tiempos de inicio de la petición y de llegada de la respuesta, el método HTTP usado, la URL pedida, y el código y descripción de la respuesta. Las retransmisiones, peticiones sin respuesta o respuestas sin una petición previa son descartadas. El funcionamiento de este programa es el siguiente: en primer lugar, las tareas *Map* generan un par *(clave, valor)* para cada petición o respuesta HTTP. Las claves constan de las direcciones IP y los puertos tanto del cliente como del servidor, lo que permitirá disponer, en la última etapa de *Reduce*, de cada petición y su respuesta en el mismo grupo, como se explicó en la sección 4.3. Los valores que generan las tareas *Map* están formados por el resto de información importante para generar la salida final: el número *ACK*, el siguiente número de secuencia, el *timestamp*, la URL de la petición, el método HTTP usado y el código y descripción de la respuesta. Finalmente, las tareas *Reduce* del disector ordenan estos valores, generando una línea del fichero de salida por cada conexión HTTP con la información descrita anteriormente. Como curiosidad, esta salida ocupa en torno a 400 veces menos que los archivos PCAP.

Muchas de las peticiones HTTP capturadas no contienen el campo *Host* debido al truncado de los paquetes en la fase de captura. Consultar el nombre de todos los servidores mediante un script que acceda al DNS de la UAM es poco eficiente, y además dado el número de consultas por segundo a resolver, podría ser considerado un ataque de denegación de servicio, causando un posible bloqueo temporal por parte del servidor. La solución propuesta es una búsqueda en el tráfico DNS disponible para identificar el

nombre de los servidores a partir de su dirección IP. El objetivo de este disector es crear una tabla con todos los nombres consultados hasta el momento, junto con su resolución de la dirección IP. No se tendrán en cuenta, sin embargo, las direcciones IP de las máquinas que realizan las peticiones ya que, como se mencionó en la sección 3.4, estos equipos están detrás de un servidor de DNS. El programa hace uso de la clase `DnsPcapReader` explicada anteriormente, gracias a la cual las tareas *Map* reciben cada consulta DNS junto a todas las direcciones IP de la correspondiente respuesta. Las claves generadas por estas tareas son los pares (petición, respuesta), de forma que las tareas *Reduce* únicamente se utilizan para eliminar los duplicados.

Por último, el objetivo del disector de tráfico HTTPS es generar un listado con las peticiones realizadas sobre este protocolo, que contenga la siguiente información: dirección IP del cliente, nombre y dirección IP del servidor y *timestamp* de la petición. Esta es una tarea sencilla, pues las tareas *Map* reciben toda esta información directamente al leer los archivos PCAP mediante la clase `HttpsPcapReader`. No obstante se ha detectado que en muchas ocasiones, cuando se inicia una conexión HTTPS, se realizan varias peticiones al servidor contra diferentes puertos consecutivos. Por ello, las tareas *Reduce* se encargarán de eliminar estos duplicados. Esto se consigue programando las tareas *Map* para que la clave esté compuesta por las direcciones IP del cliente y del servidor, así como el nombre del servidor. Las tareas *Reduce* se encargarán de ordenar las peticiones a los mismos servidores por tiempo, e ignorar las que estén muy seguidas. El umbral de tiempo se ha establecido en 1 segundo. Este intervalo es suficientemente amplio para ignorar tanto retransmisiones como el caso anómalo mencionado, y suficientemente pequeño como para tener en cuenta otros factores como el refresco automático de las páginas, que suele producirse cada decenas de segundos.

5.4. Análisis de los datos

La tercera y última parte de este proyecto, el análisis de los datos, es muy experimental, por lo que la mayor parte del trabajo realizado se contará en forma de resultados obtenidos, en el capítulo 6. No obstante, se han implementado varias clases en *Java* con el fin de probar algunas tecnologías de *Hive*.

En primer lugar, se ha creado un deserializador que transforma los paquetes en filas de una tabla. *Hive* permite hacer esto con solo implementar una interfaz, denominada `Deserializer`. Mediante su uso, los archivos en formato PCAP se conciben como una única tabla, sobre la que se podrá consultar información mediante el lenguaje *HiveQL*.

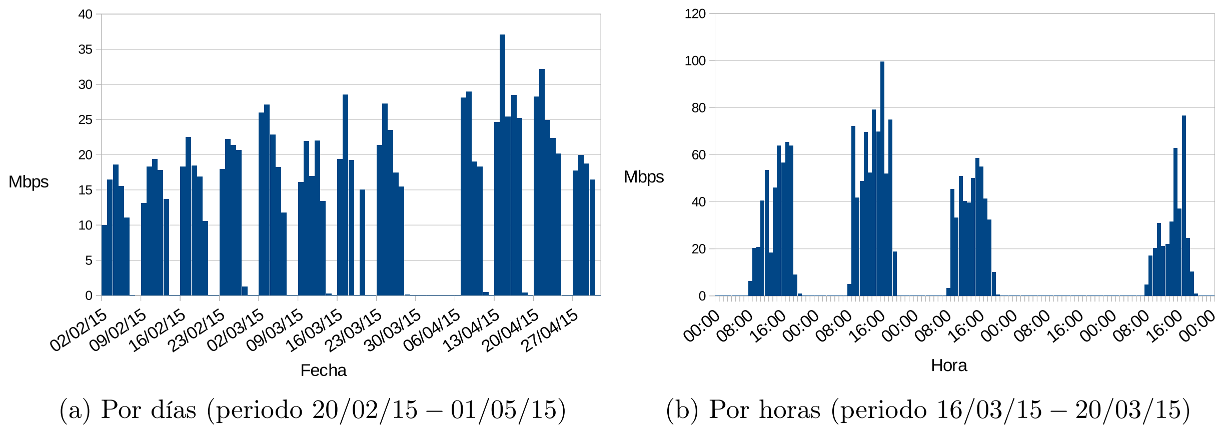
También se ha querido probar las llamadas UDF (*User Defined Functions*), que permiten crear nuevas operaciones en *Hive*. Por ejemplo, se han creado dos funciones que, a partir de un mes y un año, devuelvan, respectivamente, el curso y el semestre académicos. Con esto se podrán realizar consultas sobre los datos de un cuatrimestre específico.

6
Experimentos

El objetivo de este capítulo es mostrar los posibles usos de los datos que se han generado hasta el momento, es decir, de los archivos en formato PCAP, registros de flujos, conexiones HTTP, consultas DNS y peticiones HTTPS. Para ello se han utilizado mayoritariamente las herramientas *Hive* [20], que permite definir consultas distribuidas en un lenguaje similar a SQL, y *Hue* [33], que posibilita la gestión y ejecución de las consultas mediante una interfaz interactiva e intuitiva. Para realizar un análisis más sofisticado de los datos, se intentará reconocer patrones en el comportamiento de los estudiantes. Al intentar lograr este objetivo, se han encontrado dos problemas principales: diferenciar las peticiones HTTP realizadas directamente por los usuarios de las secundarias generadas por el navegador, e identificar los momentos en los que hay un cambio de usuario en los ordenadores. Para ello, se utilizarán herramientas como Matlab [24] y Weka [26] para generar gráficas y realizar aprendizaje automático sobre los datos.

6.1. Análisis mediante Hive

En esta sección se proponen algunas de las posibles consultas que pueden realizarse sobre los datos disponibles. Estos ejemplos servirán de base para que el administrador de la red pueda crear algunos nuevos, o modificar los que ya tiene en función de sus necesidades. Las consultas se han guardado en una interfaz web que provee la herramienta *Hue* [33] de *Cloudera*, la cual permite crear nuevas, modificar las ya existentes o ejecutarlas. Esto simplifica significativamente la labor del administrador, pues no tiene que preocuparse por el proceso de obtención de los datos, solamente de crear las consultas apropiadas. Las consultas en *HiveQL* son traducidas de forma automática en uno o varios programas MapReduce consecutivos. Por ello, la computación sigue siendo distribuida, aunque de



(a) Por días (periodo 20/02/15 – 01/05/15) (b) Por horas (periodo 16/03/15 – 20/03/15)

Figura 6.1: Series temporales del consumo de la red

forma transparente para el programador.

En primer lugar, se dispone de las tablas *flujos*, *HTTP*, *DNS* y *HTTPS*, rellenas con la salida de la herramienta *FlowProcess* y de los disectores implementados. Se ha creado también la tabla *PCAPS* con la información de los archivos en formato PCAP, mediante el deserializador comentado en la sección 5.4. Otras dos tablas importantes para la posterior elaboración de consultas son *laboratorios* y *horarios*. Estas contienen, respectivamente, la relación entre los laboratorios docentes y sus direcciones IP, y las asignaturas que se imparten en cada momento en los diferentes laboratorios.

Las consultas más sencillas se pueden realizar a partir de los ficheros de flujos. Ejemplos claros de ello son las series temporales del ancho de banda consumido. En la Figura 6.1 se pueden observar dos de estas series. En primer lugar, en la Figura 6.1a se muestra el consumo medio de la red durante el segundo cuatrimestre del curso 2014/2015, con granularidad diaria. Se aprecia un claro patrón semanal en el que, salvo excepciones, dos días sin apenas tráfico (fines de semana) son seguidos de cinco con una alta actividad (días lectivos). No obstante, existen algunas jornadas que no cumplen dicho patrón. Estos se corresponden con la Semana Santa (desde el 28 de marzo hasta el 6 de abril) y otros días festivos (27 de febrero, fiesta de la EPS, 19 de marzo, Día del Padre y 1 de mayo, Día del Trabajador). Por otro lado, en la Figura 6.1b se ha representado el mismo concepto, pero con una mayor granularidad: se puede distinguir un patrón diario, en el que la actividad comienza aproximadamente a las 8:00 y finaliza sobre las 20:00. Se ha querido incluir también un día festivo, como lo fue el 19 de marzo, para mostrar la ausencia de actividad alguna durante el mismo.

Con los ficheros de flujos también se pueden estudiar los protocolos y puertos utilizados por los ordenadores. Esto podría aprovecharse, por ejemplo, para detectar si el cortafuegos está funcionando correctamente. En la Figura 6.2 se representan los protocolos y puertos más utilizados en la red de los laboratorios. Se percibe un claro predominio de los puertos 80 y 443 sobre TCP, que suelen corresponderse con los protocolos HTTP y HTTPS. Se puede observar a su vez que tanto en el caso de estos dos protocolos, como en el de SSH (puerto 22 sobre TCP), cada flujo contiene muchos más bytes de lo normal. Esto

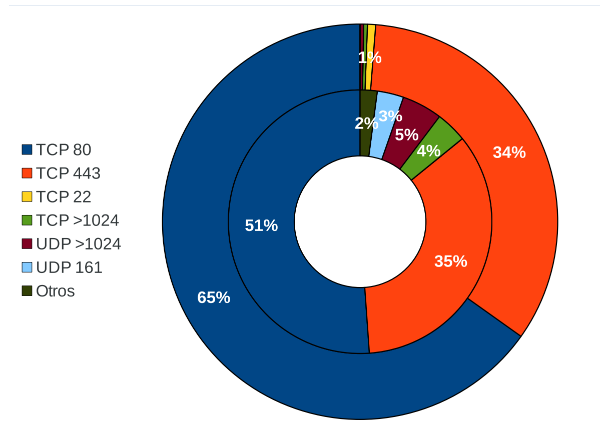
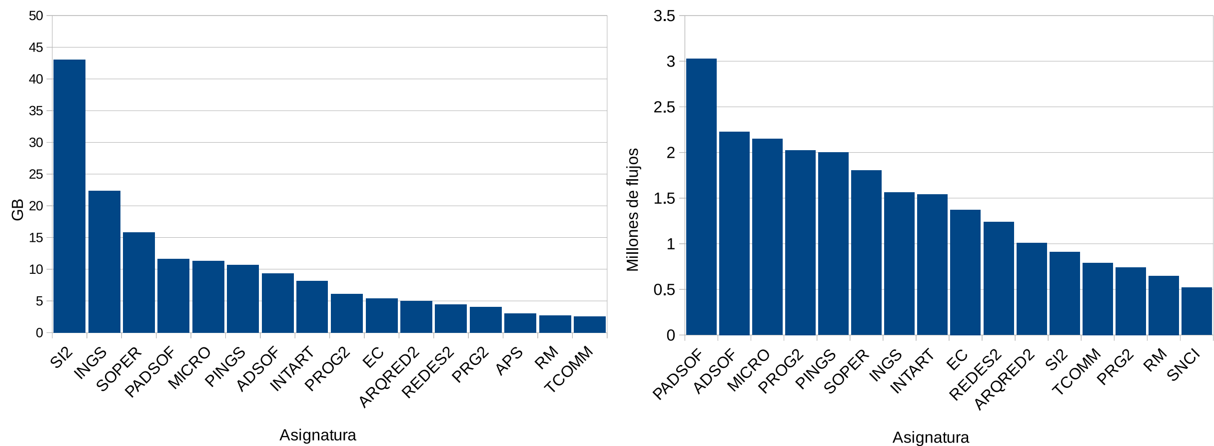


Figura 6.2: Bytes (exterior) y flujos (interior) usados por cada protocolo y puerto

es razonable, puesto que los tres suelen ser utilizados para el envío de archivos pesados. Por otro lado, el protocolo SNMP (puerto 161 sobre UDP), utilizado para intercambiar información de administración entre dispositivos de red, se caracteriza por tener flujos muy pequeños, lo cual se refleja claramente en la figura mencionada.



(a) Consumo en gigabytes.

(b) Consumo en millones de flujos.

Figura 6.3: Las 16 asignaturas que más han consumido (periodo 02/02/15 – 01/05/15)

Otro ejemplo de consultas que hacen uso de la tabla de flujos, además de las de horarios y laboratorios, es el análisis del consumo de red durante las clases prácticas de cada asignatura (ver Figura 6.3). Esto podría usarse para optimizar los recursos de la red y mejorar así la experiencia de sus usuarios. Por ejemplo, se pueden crear horarios que eviten el solapamiento de las asignaturas que más tráfico generan.

Una consulta sencilla, aunque muy útil, puede detectar las máquinas que se quedan encendidas durante la noche. Por motivos de confidencialidad, no se publicarán las direcciones IP. No obstante, en la Figura 6.4, se muestra la evolución del número de ordenadores encendidos durante las noches de un cuatrimestre. Se puede apreciar cómo la noche del viernes 6 de febrero se quedaron encendidos cientos de equipos, probablemente

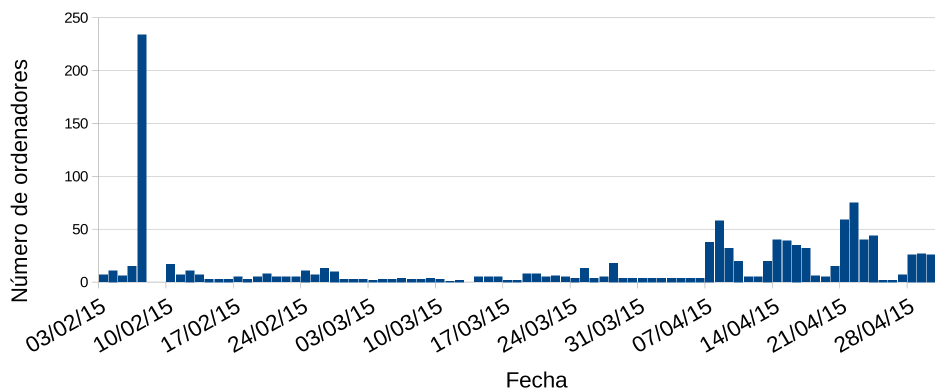


Figura 6.4: Equipos encendidos cada noche del curso (periodo 02/02/15 – 01/05/15)

por motivos de instalación de *software* o configuración de los mismos. Sin embargo, el fin de semana posterior a esta configuración, todas las máquinas quedaron apagadas, que como podemos observar en el resto del periodo, es algo bastante inusual. Por otra parte, durante las últimas semanas de curso, los alumnos dejaron un mayor número de ordenadores encendidos.

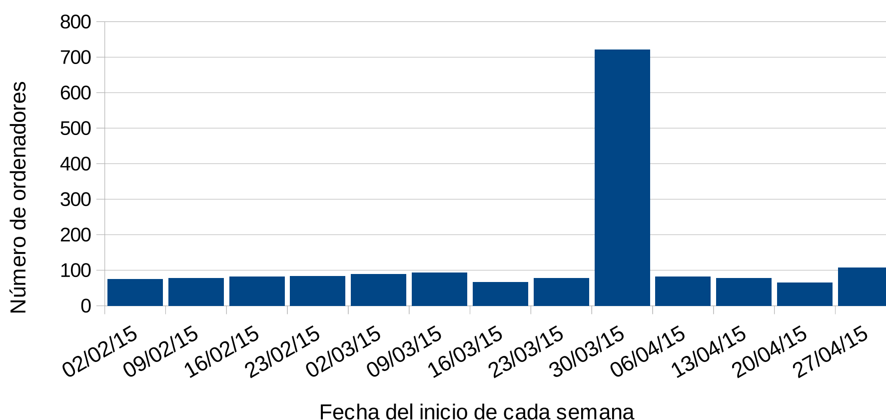


Figura 6.5: Equipos inactivos cada semana del curso (periodo 02/02/15 – 01/05/15)

Tampoco es difícil detectar los equipos que llevan sin usarse varios días seguidos, lo que puede facilitar la labor de encontrar aquellos que están averiados. En la Figura 6.5 se ha representado el número de ordenadores inactivos durante cada semana de un cuatrimestre. Cabe destacar que 46 máquinas han permanecido inactivas durante todo este periodo. Sin embargo, un promedio de 82 máquinas por semana no se han utilizado, exceptuando, claro está, la Semana Santa (30 de marzo - 5 de abril), en la que permanecieron inactivas más de 700. Tanto para esta consulta como para la anterior, se han utilizado las UDFs creadas y mencionadas en la sección 5.4, que permiten consultar los datos de un cuatrimestre específico.

Por otro lado, mediante el uso de las relaciones HTTP y HTTPS de la base de datos, se pueden obtener las páginas web más visitadas (ver Tabla 6.1). En la Tabla 6.1a se puede observar un predominio de páginas destinadas a proporcionar imágenes y elementos

Top	Host	Peticiones	Host	Peticiones
1	www.uam.es	11736207	moodle.uam.es	362888
2	ib.adnxs.com	1231283	webmail.uam.es	307115
3	pagead2.googlesyndication.com	529345	ssl.gstatic.com	285741
4	estaticos.marca.com	430052	www.google.com	255860
5	as01.epimg.net	379228	clients2.google.com	239056
6	www.discoverymax.marca.com	367725	safebrowsing.google.com	221728
7	estaticos03.marca.com	354082	www.google.es	215820
8	ts6.travian.net	334873	www.gstatic.com	210488
9	videos3.lasteles.com	332998	apis.google.com	202893
10	estaticos01.marca.com	324090	tools.google.com	163153

(a) Páginas HTTP (b) Páginas HTTPS

Tabla 6.1: Páginas web más solicitadas vía HTTP y HTTPS

estáticos, como los elementos 4 y 5 de la lista, así como páginas de anuncios, como los elementos 2 y 3. Cabe destacar que la mayoría de estas peticiones no son realizadas por los usuarios directamente. En la siguiente sección se tratará esto con mayor detalle.

6.2. Tiempo entre peticiones

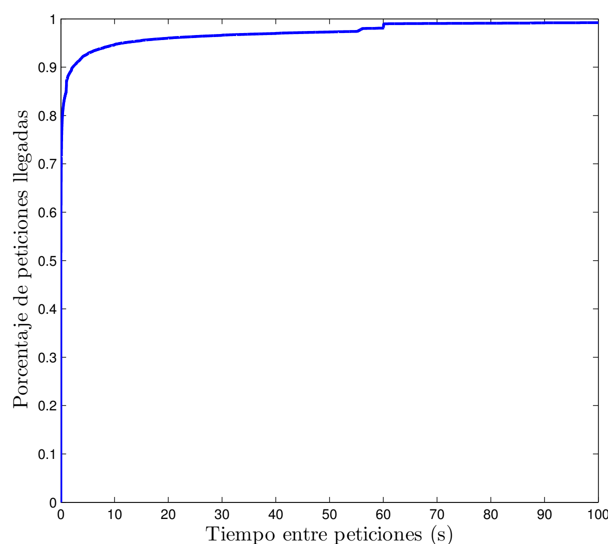
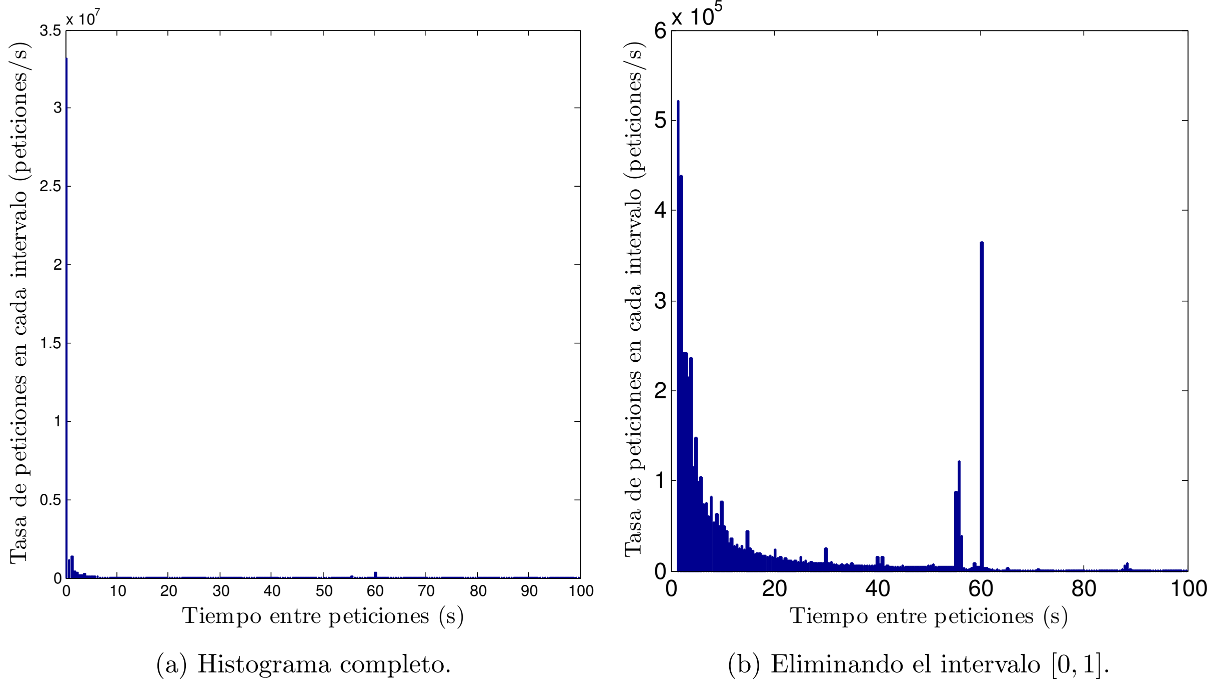


Figura 6.6: ECDF del tiempo entre peticiones

Para diferenciar las solicitudes de páginas web realizadas por los usuarios de las secundarias, realizadas por el navegador, se ha estudiado el tiempo transcurrido entre las peticiones HTTP consecutivas. Entendemos cómo petición secundaria aquellas incluidas en el documento HTML de la solicitud inicial, cómo pueden ser las imágenes, archivos *Javascript* o publicidad. Como se ha mostrado en la sección anterior, muchas de las



(a) Histograma completo.

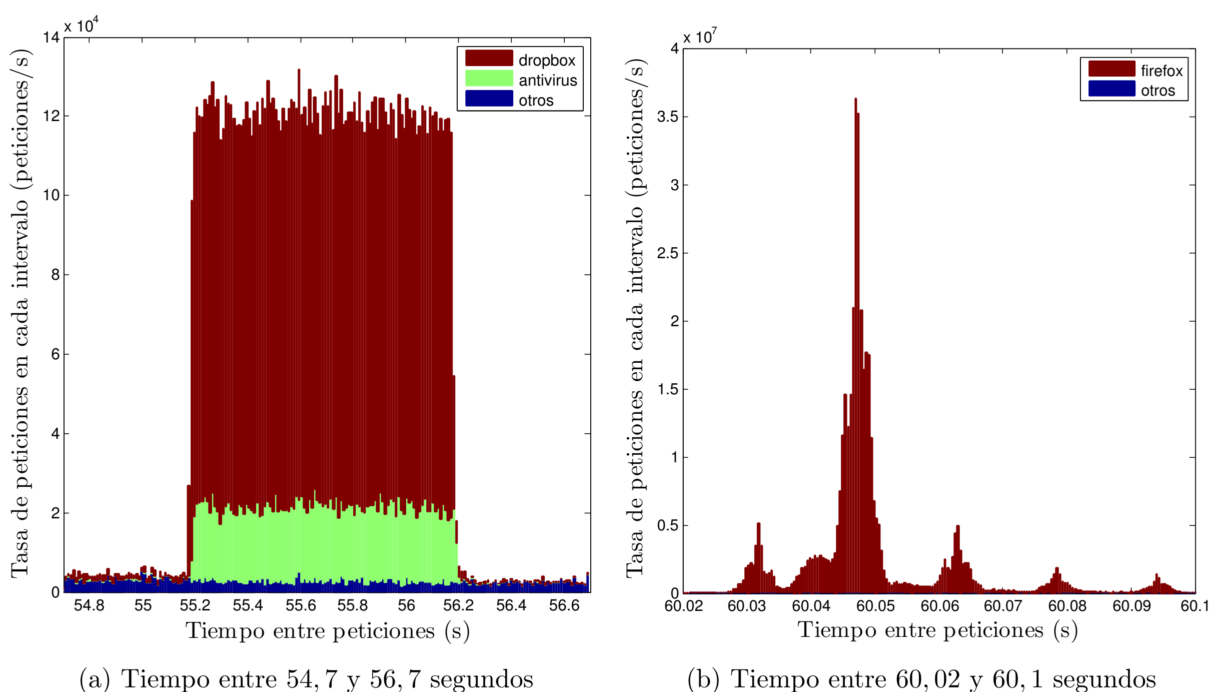
(b) Eliminando el intervalo $[0, 1]$.

Figura 6.7: Histogramas del tiempo entre peticiones

peticiones HTTP son hacia páginas exclusivamente dedicadas a ofrecer este tipo de servicios. Por ello, un filtrado de este tipo de registros reflejará mejor el comportamiento de los estudiantes. Para la generación de todas las gráficas de esta sección se ha utilizado *Matlab* [24], capaz de tratar ficheros que ocupan cientos de megabytes, como es el caso de los que se disponen.

En primer lugar, se ha dibujado una ECDF del tiempo entre las peticiones HTTP, que permite calcular el porcentaje de solicitudes realizadas antes de un tiempo determinado. En la Figura 6.6 se puede comprobar que durante el primer segundo se efectúan aproximadamente el 85 % de las peticiones. Como el usuario medio suele tardar bastante más tiempo en reaccionar y navegar por la web, se puede considerar que todas las solicitudes realizadas antes de un segundo son secundarias. En la Figura 6.7 se muestra, mediante dos histogramas, la diferencia entre incluir o no estas peticiones.

Gracias a este estudio se han detectado varias anomalías: en la Figura 6.7b pueden observarse dos picos que se sitúan entre los 50 y los 60 segundos. Tras investigar las direcciones de destino de estas peticiones, se han identificado tres claras causas de estas anomalías, que se han representado en la Figura 6.8. La primera irregularidad, tiene lugar entre los segundos $55, 2$ y $56, 2$ (ver Figura 6.8a). Sus causas son las actualizaciones automáticas de la página web de *Dropbox* y, en menor medida, la página del antivirus de la UAM. Resulta curioso que las peticiones no se concentran en un punto, si no que se distribuyen uniformemente en este periodo de un segundo. La segunda singularidad se produce, aproximadamente, en el segundo $60, 04$ (ver Figura 6.8b). Esta se debe a las actualizaciones automáticas del navegador *Firefox*, que está programado para realizar peticiones una vez por minuto.



(a) Tiempo entre $54,7$ y $56,7$ segundos

(b) Tiempo entre $60,02$ y $60,1$ segundos

Figura 6.8: Histogramas del tiempo entre peticiones, clasificados por la URL

6.3. Predicciones

Un objetivo ambicioso de este trabajo es la predicción de patrones de comportamiento de los usuarios. Para lograr este propósito, se deben conocer en primer lugar los momentos en los que los estudiantes inician y cierran sesión. Sin embargo, por motivos de privacidad no se dispone del tráfico LDAP, que contiene datos como los nombres de los usuarios y el momento en el que inician y cierran sesión en una máquina. Por este motivo, se han examinado los registros de las conexiones HTTP, junto al tiempo entre las conexiones consecutivas explicado en la sección anterior, para crear un algoritmo que reconstruya las sesiones de los usuarios. Cada sesión constará del momento de inicio y finalización de la misma, así como del sistema operativo en el que se inició la máquina.

Un simple comando de *awk* puede filtrar todas las peticiones HTTP que sucedieron a un largo periodo de inactividad. Con esto se pretende encontrar qué peticiones son realizadas automáticamente por los equipos de los laboratorios al ser arrancados. Como era de esperar, se identifican dos tipos de conexiones de arranque: un primer tipo, característico del sistema operativo *Windows*, solicita la URL `www.msftncsi.com/ncsi.txt`. Este es el sistema NCSI (*Network Connectivity Status Indicator*) que utiliza *Windows* para examinar la conectividad de la máquina a Internet durante el arranque. Un segundo tipo de conexiones, característico del sistema operativo *Ubuntu*, solicita la URL `geoip.ubuntu.com/lookup`. Esa petición devuelve información sobre la IP de la máquina y su localización aproximada. Investigando algo más se puede comprobar que ambas peticiones se realizan únicamente durante el arranque de los ordenadores, pues siempre aparecen seguidas de largos periodos de inactividad.

Con lo realizado hasta ahora se pueden detectar los momentos en los que se encienden las máquinas. Generalmente, los estudiantes apagan los ordenadores al terminar sus tareas por motivos de seguridad, ya que de esta forma todos los datos del usuario son borrados. Por tanto, el método anterior funciona bastante bien para detectar los momentos iniciales y finales de cada sesión. Sin embargo, como ya se vio en la sección 6.1, en algunas ocasiones los ordenadores se dejan encendidos, bien a propósito, bien por descuidos. Además, es muy común el caso de encender varias veces consecutivamente una misma máquina, por error en la elección inicial del sistema operativo. Para lidiar con estos casos no contemplados anteriormente, se han incluido algunas mejoras en el algoritmo de reconstrucción de sesiones:

- Las sesiones de los estudiantes expirarán tras un periodo de inactividad superior a 8000 segundos (algo más de dos horas). Con esto, las máquinas que se queden encendidas sin ningún tipo de actividad de red, se detectarán tras este periodo. Este umbral de tiempo no se puede reducir más, ya que es posible que los estudiantes utilicen el navegador de forma intermitente, con largos intervalos de inactividad intercalados.
- El tiempo de inicio de dos sesiones consecutivas no puede ser inferior a 5 minutos. Este umbral evitará la creación de diferentes sesiones si se arranca una máquina varias veces seguidas, como consecuencia de una elección equivocada del sistema operativo.

El algoritmo descrito se ha implementado en *Python* [25], puesto que el conjunto de entrada no es suficientemente grande para que se aprovechen las capacidades de *Hadoop*. Además, el lenguaje *HiveQL* no es tan versátil como *Python*, por lo que se reduce el tiempo necesario para su codificación.

La comprobación del correcto funcionamiento de la heurística explicada es una tarea difícil, pues no se conoce, a priori, las sesiones reales. No obstante, el algoritmo se utilizará como medio para predecir los momentos en los que hay clase en cada laboratorio. Como sí que se dispone de esta información, una precisa predicción sería un buen indicador de que la heurística es correcta. Con el objetivo de utilizar una herramienta de aprendizaje automático, se ha obtenido, para cada día de la semana, hora lectiva (de 9:00 a 20:00) y laboratorio, un vector con información resumida de las sesiones abiertas en esos periodos. Este vector contiene el número total de sesiones; la media, mediana y varianza de los tiempos de inicio de las sesiones con respecto al inicio de la hora; la media, mediana y varianza de las duraciones de las sesiones; y el porcentaje de sesiones que se iniciaron en *Ubuntu*. A cada hora se le ha asignado una de las siguientes clases: Clase 1, si comienza una lección de una hora; Clase 2, si comienza una lección de dos horas; y Clase 0, si no comienza ninguna lección en esa hora. En un principio se tuvo en cuenta una cuarta clase, la Clase 3, que representaba las horas en las que, no empezando ninguna lección, había clase (por ejemplo, la segunda hora de una lección que dura dos horas). Esto se hizo para evitar problemas de confusión de esta Clase 3 con la Clase 0. Sin embargo, tras examinar los resultados, se comprobó que la Clase 3 se clasificaba siempre como Clase 0. Por ello, se decidió unificar estas dos clases.

Una vez hecha esta clasificación, se puede utilizar la herramienta *Weka* [26] para realizar un aprendizaje automático supervisado. Dado que el conjunto de entrada es muy reducido (1100 entradas), esta herramienta es idónea y no se necesitarán técnicas Big Data. Las mejores predicciones se han obtenido utilizando el algoritmo *Random Forest*. En la Tabla 6.2 se muestra la matriz de confusión resultante. Se puede observar que el porcentaje de aciertos para las clases 0, 1 y 2 son, respectivamente, 98, 25 %, 34, 62 % y 88, 13 %. El bajo número de aciertos de la Clase 1 podría deberse a la reducida cantidad de elementos de la misma (solo un 2, 4 %). Se obtiene, por tanto, una tasa de aciertos global del 95, 27 %, que es bastante satisfactoria. Se concluye así que la heurística creada para detectar las sesiones de los usuarios cumple con los objetivos propuestos.

Clase 0	Clase 1	Clase 2	Predicción Realizada
898	2	14	Clase 0
13	9	4	Clase 1
18	1	141	Clase 2

Tabla 6.2: Matriz de confusión de la predicción de horarios

7
Pruebas

En este capítulo se detallan las pruebas realizadas del sistema implementado. Se muestran las características de los sistemas de pruebas, y algunas gráficas que comparan el rendimiento alcanzado frente al de otras soluciones existentes en el mercado. Por último se explican las pruebas de verificación y validación realizadas.

7.1. Pruebas de rendimiento

Una de las primeras motivaciones de este trabajo era conocer hasta qué punto *Hadoop* representa una alternativa a los servidores de alto rendimiento. Para ello se ha comparado el rendimiento de tres versiones del disector HTTP, que corresponde con el protocolo más común en los datos disponibles: una implementación en *C* [10], la versión que hace uso de MapReduce presentada anteriormente (ver sección 5.3), y esta misma implementación, serializada en *Java*. Esta prueba se ha realizado sobre un servidor de alto rendimiento (a partir de ahora SAR) con 2 procesadores *Intel Xeon E5-2630* @ $2,6$ GHz (12 cores en total), 32 GB de RAM y RAID 0 HW de 9 discos de 3 TB. Este equipo permite realizar una lectura sostenida de los datos a una tasa de 10 Gbps.

La prueba ha consistido en procesar un conjunto de ficheros PCAP de 1 GB, para un total de datos que varía desde 1 GB (1 solo fichero) hasta 1 TB (1024 ficheros), guardados en el RAID. Para las pruebas con *Hadoop*, se ha realizado una instalación pseudo-distribuida (un solo nodo) en el propio servidor. Se ha configurado para que las tareas MapReduce hagan uso de los 12 cores disponibles, y que HDFS lea del RAID.

En la Figura 7.1 se muestran los resultados obtenidos. Como era de esperar, la versión en *C* presenta un rendimiento superior a la implementación en *Java*. Esto permite estimar

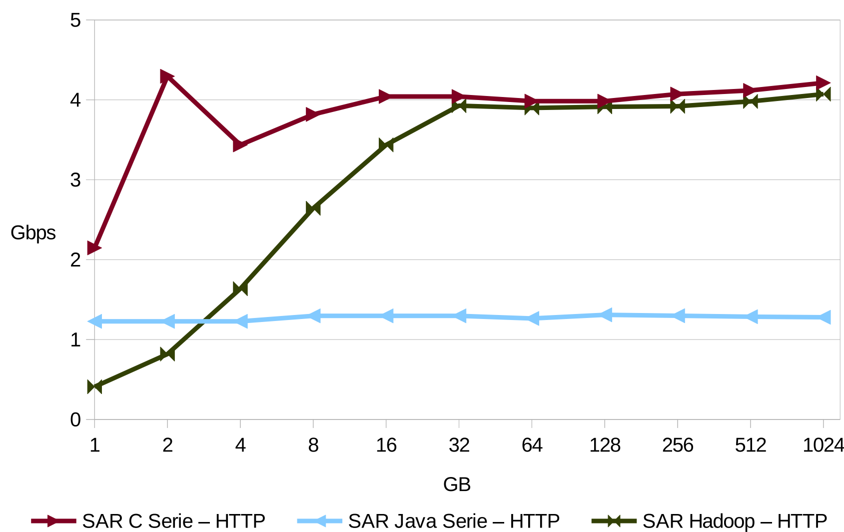


Figura 7.1: Rendimiento de las tres versiones del disector de HTTP en SAR

una de las desventajas iniciales de *Hadoop* frente a *C*. Cabe destacar que la implementación *C* supera por muy poco los 4 Gbps, mientras que en escritura secuencial sostenida el RAID aguanta 10 Gbps, por lo que, el cuello de botella no es el disco.

Los resultados de la implementación en *Hadoop* son interesantes: el rendimiento obtenido al procesar un solo fichero es mucho peor que la versión de *Java* en serie. Esto se debe a los diferentes *overheads* de *Hadoop*: comunicación, sincronización, planificación de tareas, etc. y a que los datos están en HDFS, una capa por encima del sistema de ficheros del RAID. A medida que se aumenta el número de ficheros, entran en juego más tareas *Map*, ya que cada core puede procesar un único archivo. Es a partir de aproximadamente 3 cores cuando el *overhead* de *Hadoop* ya no supone un problema, y se mejora el rendimiento de la versión *Java*. El número de cores usados continúa incrementándose hasta los 12 ficheros. A partir de entonces comienzan a asignarse varias tareas a cada core. Sin embargo, el rendimiento sigue aumentando hasta los 32 archivos, momento en el cual se estabiliza. Por lo tanto, hacen falta más de dos tareas por cada core para alcanzar el máximo rendimiento, el cual es ligeramente inferior al de la versión de *C*. Adicionalmente, si se utilizasen los nueve discos por separado, es de esperar que el rendimiento de *Hadoop* mejorase. Esto se debe a que la lectura del RAID se realiza a la velocidad del disco más lento, que evidentemente, es menor que la velocidad media que se alcanzaría mediante un acceso paralelo a los discos por separado.

7.1.1. Clúster Hadoop

Una vez analizado el rendimiento en un servidor de altas prestaciones, el nuevo objetivo es evaluar el rendimiento en el clúster *Hadoop*. Este sistema consta de 8 nodos conectados mediante una red de 1 Gbps, de los cuales solo 5 son utilizados para la computación y el almacenamiento. La Tabla 7.1 muestra los detalles de estos equipos, así como del servidor de alto rendimiento (SAR) empleado en las pruebas anteriores. Esta tabla también incluye

las características de los clústeres empleados por Y. Lee [12] para realizar sus pruebas de rendimiento, dado que más adelante se realizará una comparativa con su trabajo.

Tabla 7.1: Características de los sistemas de pruebas

Sistema		RAM (GB)	Disco (TB)	CPU	Total cores	Red (Gbps)
Hadoop	Esclavo 1	32	15,2	1x Xeon L5408 @ 2,13 GHz	4	1
	Esclavo 2	32	15,2	1x Xeon L5408 @ 2,13 GHz	4	1
	Esclavo 3	256	12,1	4x Xeon E7-4830 @ 2,13 GHz	32	1
	Esclavo 4	64	20,6	2x Xeon E5-2620 v3 @ 2,40 GHz	12	1
	Esclavo 5	64	20,6	2x Xeon E5-2620 v3 @ 2,40 GHz	12	1
SAR		32	RAID 0: 9x 3 TB	2x Xeon E5-2630 @ 2,6 GHz	12	-
Y. Lee [12] (30 nodos)		30x 19	30x 40	30x 8 cores @ 2,93 GHz	240	1

El primer paso para estimar el rendimiento del clúster ha sido ejecutar `TestDFSIO`, un *benchmark* de E/S distribuido incluido en *Hadoop* que permite evaluar el rendimiento de lectura de HDFS. La Figura 7.2 muestra la tasa de lectura obtenida al leer entre 1 y 1024 ficheros de 1 GB generados por la misma aplicación. Resultados similares se obtienen si se ejecuta el *benchmark* sobre el mismo número de archivos PCAP, pues este no tiene en cuenta el contenido de los ficheros, y solo afecta al rendimiento su tamaño y distribución por el clúster. Se puede observar que la tasa de lectura se incrementa con el número de ficheros y no llega a saturar. Sería necesario probar con una cantidad de datos superior para conocer los límites del clúster, seguramente al llegar a la tasa soportada por los discos o por la red.

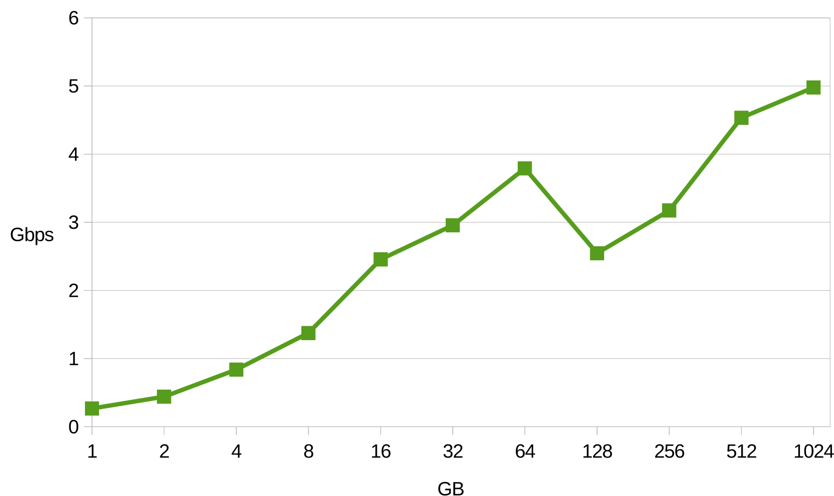


Figura 7.2: Rendimiento del *test* de lectura en *Hadoop*

Cabe destacar el cambio de comportamiento al alcanzar los 64 GB (o 64 ficheros), el cual se muestra en detalle en la Figura 7.3. La bajada de la tasa de lectura global se debe a que el nodo más potente consume todos los datos locales y comienza a solicitar archivos a otros nodos, afectando al rendimiento. Posteriormente, la tasa de lectura sigue mejorando, ya que la red no está saturada todavía. En resumen, se pasa de procesar datos locales a procesar datos remotos. Dada la heterogeneidad del clúster en cuanto a número

de cores, la potencia de cálculo de los cores y el número de discos por nodo, sería necesario un estudio más detallado para obtener una conclusión más concreta.

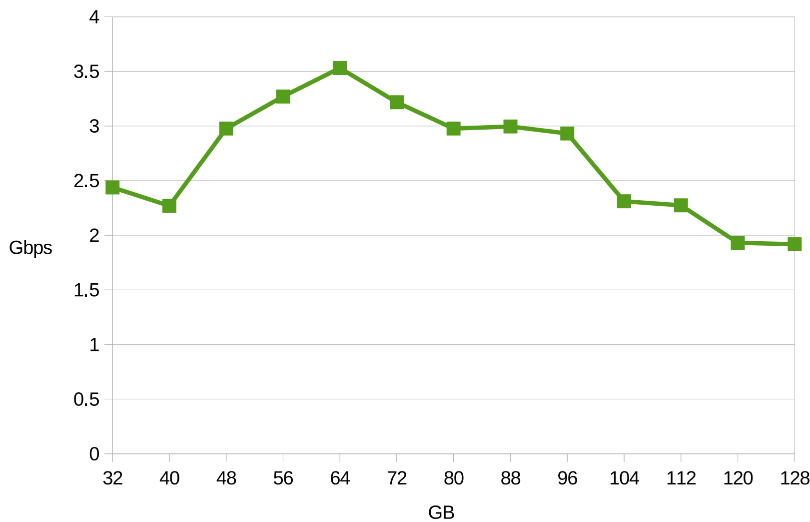


Figura 7.3: Rendimiento del *test* de lectura en *Hadoop* con mayor granularidad

Una vez realizada la prueba de lectura, se ha procedido a ejecutar los diferentes disectores realizados en el clúster de *Hadoop*. La Figura 7.4 muestra los rendimientos obtenidos por estas tres aplicaciones, junto al del *test* de lectura anterior. Todas estas pruebas se han realizado sobre el mismo conjunto de ficheros de entrada.

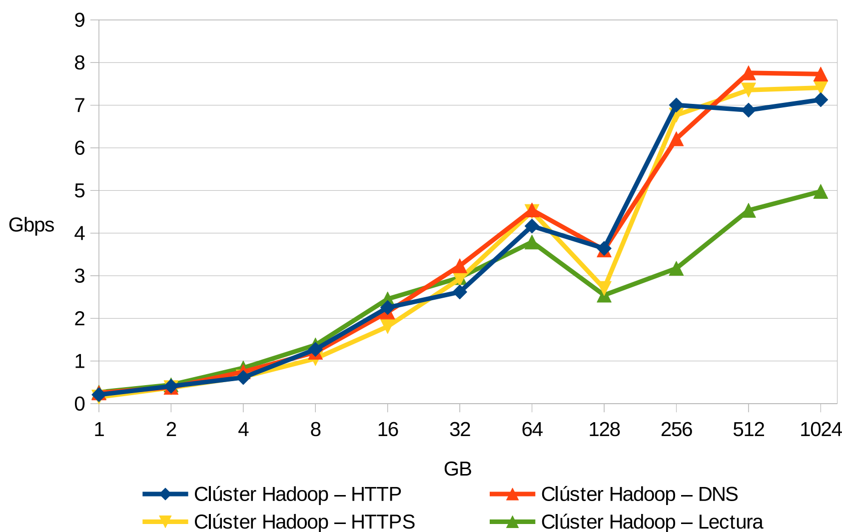


Figura 7.4: Rendimiento de los disectores y el *test* de lectura en el clúster de *Hadoop*

Se puede apreciar que, al contrario de lo esperado, el rendimiento de los tres disectores es superior al del *test*. La causa de este fenómeno es que, mientras el *test* permite la división de los ficheros en bloques de 128 MB, los disectores no (ver sección 5.1). Por tanto, el *test* debe lanzar 8 veces más tareas, lo que supone una sobrecarga tanto a la hora de planificarlas como al crear y destruir los procesos *Java* correspondientes.

En la figura mencionada, se puede también observar que para obtener el máximo rendimiento hace falta procesar un número de ficheros superior a cuatro veces el número de cores disponibles. Por otro lado, en las pruebas en SAR eran necesarios tres veces más. Esto se debe a que la sobrecarga causada por la planificación, comunicación y sincronización es mucho mayor en el clúster de *Hadoop*, por lo se necesita una mayor cantidad de datos para aprovechar completamente las características de *Hadoop*.

A su vez, se puede observar que el rendimiento de los diferentes disectores es muy parecido, a pesar de que el procesamiento de cada uno de ellos es distinto. Un dato relevante es que la cantidad de tráfico DNS es varios órdenes de magnitud inferior al tráfico HTTP o HTTPS, y no por ello el disector correspondiente tarda mucho menos. Esto es debido a que mayor parte del tiempo se consume en la propia lectura de los ficheros, pues es necesario leer todos y cada uno de los paquetes aunque posteriormente no se realice nada con su contenido. Adicionalmente, al igual que en el *test* de lectura comentado anteriormente, se percibe una caída del rendimiento desde los 64 hasta los 128 GB. Su causa sigue siendo la heterogeneidad del clúster, que obliga al nodo más potente a solicitar los datos remotos cuando acaba de procesar los archivos locales.

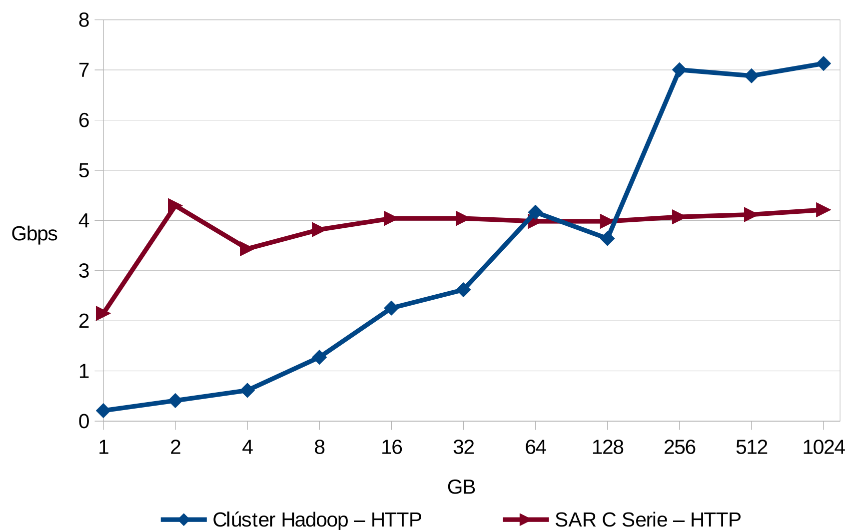


Figura 7.5: Rendimiento de los disectores de HTTP en *Hadoop* y en SAR

Por último, en la Figura 7.5 se comparan dos de los rendimientos vistos hasta ahora: la implementación en *C* del disector HTTP, ejecutada en SAR, y la versión en *Hadoop*, probada sobre el clúster. Se observa que hacen falta al menos 64 ficheros para que el clúster alcance el rendimiento de la implementación serie. Sin embargo, apenas se aumenta el rendimiento en un 70 % a pesar de utilizar una cantidad de recursos muchísimo mayor. Esto pone en entredicho que *Hadoop* represente una buena opción si se persigue el rendimiento como principal objetivo. No obstante, hay que recordar que dicho proyecto tiene otras prioridades, como lo son la escalabilidad, la fiabilidad y la transparencia de la paralelización de los algoritmos y de la distribución de las tareas. Como consecuencia de esta escalabilidad, un aumento del número de nodos implicaría un aumento proporcional del rendimiento obtenido, lo que es una gran ventaja si se dispone un clúster con cientos de equipos.

7.1.2. Comparativa de rendimiento

Para finalizar esta sección, se va a comparar el rendimiento de los disectores desarrollados frente a otras implementaciones disponibles en la literatura. En la Tabla 7.1 se detallaron las características de los sistemas utilizados en este trabajo, así como la información del clúster empleado por Y. Lee en su trabajo [12]. Es importante destacar que la comparación con las pruebas de Y. Lee no es totalmente justa, pues ni el tráfico disponible ni la funcionalidad implementada son idénticos. No obstante, todas sus aplicaciones alcanzan rendimientos parecidos, al igual que pasaba con los nuestros. En la Tabla 7.2 se muestran los rendimientos de todas estas pruebas, medidos en *Gbps*, y en *Gbps* por cada core utilizado. Se puede observar que el rendimiento de cada core del sistema implementado duplica el de los cores de Y. Lee. Al mismo tiempo, el rendimiento de la versión de Y. Lee es $3,5$ veces superior que el de la implementación de RIPE [11] de la que se partió inicialmente.

Tabla 7.2: Rendimiento de cada sistema de pruebas

Sistema	Programa	Gbps	Gbps/core
Hadoop	*Test* de lectura	4,98	0,08
	Disector HTTP	7,13	0,11
	Disector HTTPS	7,41	0,12
	Disector DNS	7,72	0,12
SAR	Disector HTTP	4,21	4,21
Y. Lee [12]	5 nodos	1,9	0,05
	30 nodos	14	0,06

7.2. Pruebas de verificación y validación

Con el objetivo de comprobar que se ha implementado el sistema correctamente, se han utilizado dos herramientas: *Wireshark* [4] y la versión en *C* del disector de tráfico HTTP [10]. La primera de ellas permite verificar que los paquetes capturados están truncados y bien formados, de modo que la salida del módulo de captura es correcto. Se ha comprobado también que el programa no pierde ningún paquete como consecuencia de los cambios realizados. Para ello, se han ejecutado simultáneamente las versiones inicial y final del programa, comprobándose posteriormente que ambas capturaron el mismo número de paquetes.

Para verificar que las clases creadas para la lectura de los protocolos HTTP, HTTPS y DNS funcionan adecuadamente, se ha creado un programa que imprime cada paquete devuelto por los iteradores de cada una de las tres clases. Posteriormente se han cotejado las salidas de dicho programa y la esperada, obtenida mediante *Wireshark*. Por último, para probar que la reconstrucción de conexiones HTTP es correcta, se ha comprobado que las salidas del disector implementado eh *Hadoop* y la de la versión en *C* son idénticas.

8 Conclusiones

A lo largo de este Trabajo de Fin de Grado se ha estudiado la viabilidad del proyecto *Apache Hadoop* en el contexto de la monitorización de las redes de comunicaciones. Se ha creado un prototipo que realiza desde una captura inicial de los paquetes de red y reconstrucción de flujos hasta un final análisis a alto nivel de los datos. Se ha podido instalar y probar esta herramienta en un entorno real como es la subred de los laboratorios docentes de la EPS.

Se ha conseguido abrir, de una forma eficiente, los archivos en formato PCAP almacenados en HDFS, así como interpretar tres protocolos fundamentales para las comunicaciones de hoy en día, como lo son HTTP, HTTPS y DNS. Aunque para alcanzar este objetivo se ha partido de la librería de RIPE [11], se han reconocido y solucionado sus principales problemas tanto de rendimiento como por incompatibilidades con el tráfico disponible. Asimismo, se han logrado identificar los cuellos de botella del clúster que justifican los resultados de rendimiento obtenidos. Se ha podido superar el rendimiento de las demás implementaciones disponibles en la literatura, consiguiendo una tasa superior a 7 Gbps, y 120 Mbps por cada core utilizado.

Por último, se ha proporcionado una batería de ejemplos de consultas en *HiveQL* para el análisis de los datos en HDFS. Estas se pueden ejecutar y/o modificar mediante la interfaz web que ofrece la herramienta *Hue* [33], lo que simplifica las tareas de los administradores de redes, al preocuparse únicamente del análisis de los datos. Además, se han descubierto anomalías en el tráfico HTTP gracias al estudio del tiempo entre las peticiones consecutivas de este protocolo. Por último, se ha creado una heurística para identificar los momentos en los que los estudiantes inician o cierran sesión en los ordenadores, así como el sistema operativo que arrancan. Esto ha permitido predecir las horas en las que ha habido clase durante el curso, mediante el uso de aprendizaje automático supervisado.

Como conclusión personal, la realización de este trabajo ha sido una experiencia enriquecedora, pues se han puesto en práctica muchos conocimientos adquiridos a lo largo del grado. En primer lugar, se han utilizado conceptos aprendidos en Arquitectura de Sistemas Paralelos, para entender la arquitectura de *Hadoop*, así como el paradigma MapReduce, los cuales guardan mucha relación con MPI y el modelo maestro-esclavo. En esta asignatura se aprendió también a realizar y representar mediciones de rendimiento. Otras dos asignaturas clave para este trabajo han sido Redes de Comunicaciones I y II, en las que se aprendió el funcionamiento de los protocolos de red, los servidores de DNS y los cortafuegos. En las asignaturas de Estructuras de Datos y Sistemas Informáticos I se asimilaron las nociones básicas de SQL, que han servido de base para aprender y optimizar las consultas creadas en *HiveQL*. Han sido también de gran utilidad los conceptos de Programación Orientada a Objetos, y del ciclo de vida del *software* aprendidos en las asignaturas de Análisis y Diseño de Software, Ingeniería del Software, y sus respectivos proyectos. Adicionalmente, los conceptos de escalabilidad, fiabilidad, bases de datos distribuidas, y máquinas virtuales fuenon obtenidos en Sistemas Informáticos II. Las técnicas de análisis de datos desde un punto de vista del aprendizaje automático, así como el funcionamiento del programa *Weka*, se estudiaron en Inteligencia Artificial. La herramienta *Matlab* y el lenguaje de programación *Python*, se aprendieron, respectivamente, en las asignaturas de Cálculo Numérico y Laboratorio de Cálculo. Asimismo, durante el desarrollo del trabajo se han aprendido muchos comandos, ficheros internos y funcionamiento del sistema operativo *GNU/Linux*.

9
Trabajo futuro

A la monitorización de redes y el procesamiento distribuido de datos mediante *Hadoop* le queda un largo recorrido y desarrollo para llegar a alcanzar toda la funcionalidad y versatilidad del resto de herramientas actuales. Con este objetivo en mente, se han identificado algunas mejoras del sistema que podrían considerarse en un futuro.

En primer lugar, cada uno de los disectores de tráfico obtiene únicamente los registros de uno de los protocolos de la capa de aplicación. Por ello, se hace imprescindible ejecutar las tres aplicaciones creadas para obtener todos los protocolos, lo que no es nada escalable conforme se aumente el número de protocolos a analizar. Por ello, un primer paso sería cambiar el diseño tanto de la API que lee los archivos PCAP como de los propios disectores, para unificar las tres funcionalidades en una sola ejecución. Con esto se aumentaría significativamente el rendimiento, pues solo sería necesaria una única lectura de los datos, que como se mostró, ocupa la mayor parte del tiempo de ejecución.

Otro claro paso para ampliar la funcionalidad es aumentar el repertorio de protocolos que se pueden analizar. Un ejemplo sería el protocolo SNMP, que como se vio en la sección 6.1, representa el 3 % de todos los flujos generados. Además, también se plantea como trabajo futuro dar soporte a los registros DNS de tipo *CNAME*, incrementando así la cantidad de direcciones IP que pueden ser resueltas.

Respecto a la parte del análisis de los datos, se puede ampliar la batería de consultas definidas en *HiveQL*, así como mejorar las técnicas usadas de aprendizaje automático. Ante un aumento de la cantidad de datos de entrada para realizar las predicciones, se sugiere integrar esta parte con la herramienta *Apache Mahout*.

Como último objetivo, se propone profundizar más en la evaluación de las limitaciones de la arquitectura *Hadoop*, fundamentalmente estudiando si el sistema de ficheros HDFS

puede suponer un cuello de botella futuro cuando se capture a 10 Gbps o comprobando cómo afecta la red de comunicaciones al procesamiento de grandes cantidades de datos (decenas de TB). Por otro lado, ya se están buscando soluciones para acelerar el procesamiento en *Hadoop* combinando MapReduce y aceleradores *hardware*, en particular GPUs, para el procesamiento de los datos.

Bibliografía

[1] *Apache Hadoop*. [Accedido: 19/05/2015]. URL: `http://hadoop.apache.org`.

[2] Hélder Veiga y col. «Active Traffic Monitoring for Heterogeneous Environments». En: *Networking - ICN 2005*. Ed. por Pascal Lorenz y Petre Dini. Vol. 3420. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2005, págs. 603-610.

[3] *Tcpdump*. [Accedido: 19/05/2015]. URL: `http://www.tcpdump.org/`.

[4] *Wireshark*. [Accedido: 19/05/2015]. URL: `https://www.wireshark.org/`.

[5] *CoralReef Software Suite*. [Accedido: 19/05/2015]. URL: `http://www.caida.org/tools/measurement/coralreef/`.

[6] *Cisco NetFlow*. [Accedido: 19/05/2015]. URL: `http://www.cisco.com/web/go/netflow`.

[7] Jaime J Garnica y col. «A FPGA-based scalable architecture for URL legal filtering in 100GbE networks.» En: *ReConFig*. 2012, págs. 1-6.

[8] Giorgos Vasiliadis y col. «Gnort: High Performance Network Intrusion Detection Using Graphics Processors». English. En: *Recent Advances in Intrusion Detection*. Ed. por Richard Lippmann, Engin Kirda y Ari Trachtenberg. Vol. 5230. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008, págs. 116-134.

[9] R. Leira y col. «Multimedia flow classification at 10 Gbps using acceleration techniques on commodity hardware». En: *Smart Communications in Network Technologies (SaCoNeT), 2013 International Conference on*. Vol. 03. Jun. de 2013, págs. 1-5.

[10] Carlos Gonzalo Vega Moreno. «Disección de tráfico web a alta velocidad». Tesis de lic. Universidad Autónoma de Madrid, sep. de 2014.

[11] *Large-scale PCAP Data Analysis Using Apache Hadoop*. [Accedido: 19/05/2015]. URL: `https://labs.ripe.net/Members/wnagele/large-scale-pcap-data-analysis-using-apache-hadoop`.

[12] Yeonhee Lee y Youngseok Lee. «Toward Scalable Internet Traffic Measurement and Analysis with Hadoop». En: *SIGCOMM Comput. Commun. Rev.* 43.1 (ene. de 2013), págs. 5-13. ISSN: 0146-4833.

[13] Pedro María Santiago del Río. «Internet Traffic Classification for High-Performance and Off-The-Shelf Systems». Tesis doct. Universidad Autónoma de Madrid, 2013.

[14] S. Ghemawat, H. Gobioff y S. Leung. «The Google File System». En: *ACM SIGOPS Operating Systems Review* 37.5 (dic. de 2003), págs. 29-43.

[15] K. Shvachko y col. «The Hadoop Distributed File System». En: *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*. Mayo de 2010, págs. 1-10.

[16] Vinod Kumar Vavilapalli y col. «Apache hadoop YARN: yet another resource negotiator». En: *Proceedings of the 4th annual Symposium on Cloud Computing*. ACM. 2013, pág. 5.

[17] Lars George. *HBase: the definitive guide*. O'Reilly Media, Inc., 2011.

[18] Christopher Olston y col. «Pig Latin: A Not-so-foreign Language for Data Processing». En: *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*. SIGMOD '08. ACM, 2008, págs. 1099-1110.

[19] Sean Owen y col. *Mahout in action*. Manning, 2011.

[20] Ashish Thusoo y col. «Hive: a warehousing solution over a map-reduce framework». En: *Proceedings of the VLDB Endowment* 2.2 (ago. de 2009), págs. 1626-1629.

[21] *Institutions using Hadoop*. [Accedido: 19/05/2015]. URL: `https://wiki.apache.org/hadoop/PoweredBy`.

[22] *Products that include Hadoop*. [Accedido: 19/05/2015]. URL: `http://wiki.apache.org/hadoop/Distributions%20and%20Commercial%20Support`.

[23] Yeonhee Lee, Wonchul Kang y Youngseok Lee. «A Hadoop-based Packet Trace Processing Tool». En: *Proceedings of the Third International Conference on Traffic Monitoring and Analysis*. TMA'11. Springer-Verlag, 2011, págs. 51-63.

[24] *Matlab*. [Accedido: 19/05/2015]. URL: `http://www.mathworks.com/matlab`.

[25] *Python*. [Accedido: 19/05/2015]. URL: `https://www.python.org/`.

[26] Mark Hall y col. «The WEKA data mining software: an update». En: *ACM SIGKDD explorations newsletter* 11.1 (2009), págs. 10-18.

[27] Niccolò Cascarano, Luigi Ciminiera y Fulvio Risso. «Optimizing Deep Packet Inspection for High-Speed Traffic Analysis». En: *J. Netw. Syst. Manage.* 19.1 (mar. de 2011), págs. 7-31. ISSN: 1064-7570.

[28] Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 2012.

[29] T. Berners-Lee, R. Fielding y H. Frystyk. *Hypertext Transfer Protocol – HTTP/1.0*. RFC 1945. `https://www.ietf.org/rfc/rfc1945.txt`. Mayo de 1996.

[30] Roy T. Fielding y col. *Hypertext Transfer Protocol – HTTP/1.1*. RFC 2616. `https://www.ietf.org/rfc/rfc2616.txt`. Jun. de 1999.

[31] P. Mockapetris. *Domain names - implementation and specification*. RFC 1035. `https://www.ietf.org/rfc/rfc1035.txt`. Nov. de 1987.

[32] Tim Dierks y Christopher Allen. *The TLS Protocol Version 1.0*. RFC 2246. `https://www.ietf.org/rfc/rfc2246.txt`. Ene. de 1999.

[33] *Hue*. [Accedido: 19/05/2015]. URL: `http://gethue.com/`.