Trabajo de Fin de Grado

Análisis de ficheros log de la WiFi-ULL usando técnicas de Big Data

Víctor Plaza Martín

La Laguna, 8 de Septiembre de 2015

D. **Marcos Colebrook Santamaría**, con N.I.F. 43.787.808-V, Profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor.

D. **José Luis Roda García**, con N.I.F. 43.356.123-L, Profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como cotutor.

C E R T I F I C A N

Que la presente memoria titulada:

"Análisis de ficheros log de la WiFi-ULL usando técnicas de Big Data"

ha sido realizada bajo su dirección por D. **Víctor Plaza Martín**, con N.I.F. 54.057.753-W.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 7 de septiembre de 2015.

Agradecimientos

En especial a mi tutor en este proyecto, Marcos Colebrook Santamaría por esta gran experiencia, su paciencia infinita, sus ganas de enseñar, su confianza y por encima de todas las cosas, su calidad humana.

También agradecer en igual medida tanto a José Luis Roda García (cotutor) como a Carlos J. Pérez González. Al primero por su energía, pasión y ganas que contagia a los de su alrededor mientras que al segundo por su amor por el Big Data, sosiego y tranquilidad que tan necesarios fueron en muchos momentos del desarrollo.

Destacar la inestimable ayuda de José Carlos González González, jefe del Servicio TIC de la ULL, por abrirme las entrañas de su casa y nutrirnos de datos y buenos consejos a la hora de interpretar el contenido de estos ingentes ficheros de datos facilitando enormemente el trabajo.

Y por último pero no menos importante a Pedro González Yanes por soportar mi constante búsqueda del clúster ideal, muchísimas gracias por el tiempo invertido en lograrlo.

Licencia

Resumen

El objetivo de este trabajo ha sido el establecer un acercamiento gradual y con una dificultad progresiva al campo del Big Data a través de un enfoque eminentemente práctico.

La principal razón para la elección de este tema para la elaboración de este proyecto es la relevancia manifiesta que está tomando en la sociedad actual el eficiente manejo y tratamiento de unos datos cada vez más masivos y heterogéneos con el fin de obtener información útil para las organizaciones.

Para ello, desde un comienzo se estableció la posibilidad de desarrollar el proyecto en colaboración con una empresa externa al ámbito puramente académico con el fin de dotar de sentido y utilidad al conjunto del trabajo de investigación y desarrollo llevado a cabo.

Estas empresas serían las encargados de nutrirnos con datos reales y problemáticas a resolver igualmente verídicas, a la vez que nos establecerían una serie de restricciones y condiciones a la hora de trabajar.

A través del co-tutor del proyecto, José Luis Roda García, surgió la posibilidad de colaborar tanto con el Servicio Canario de Emergencias (112) como con el Servicio de TIC de la Universidad de La Laguna.

De esta manera y gracias a la ayuda de José Carlos González González, Jefe del Servicio TIC de la ULL, pudimos establecer un marco claro de trabajo con unos requisitos y restricciones definidos, unos datos de entrada verídicos y unos objetivos prácticos en las tareas que se irían realizando de manera progresiva como se verá más adelante.

Teniendo claros los objetivos y con datos sobre los que trabajar se abordó la problemática de la elección del software como solución, elementos tangenciales como puede ser el IDE de desarrollo, la madurez del lenguaje, la metodología, los requisitos de hardware necesarios y un largo etcétera.

Por último, y en colaboración con el personal del Centro de Cálculo de la ETSII se procedió a un despliegue en múltiples nodos (*multinode*) constituyendo la solución final, junto con varios ejemplos de visualización de los datos generados. También se realizaron propuestas para la automatización

del proceso de análisis de los datos con el fin de solventar algunas dificultades de uso y dotar de mayor usabilidad a la propuesta desarrollada.

Palabras clave: Big Data, Hadoop, Streaming, Python, MapReduce, Mapper, Reducer, R.

Abstract

The aim of this research project has been to establish a gradual introduction to the field of Big Data, with a progressive difficulty, using a completely practical approach.

The main reason for the topic choice of this project is the obvious relevance in today's society of an efficient manipulation and processing of data which is continuously growing and becoming more varied, with the aim of obtaining useful information for organisations.

In order to do that, from the begining there was the possibility of carrying out the project in collaboration with a company external to the academic context, for the purpose of giving sense and utility to the whole research and development project.

These companies were in charge of feeding us both real data and real problems to solve, and at the same time, they established a series of restrictions and conditions to work with.

Through the co-tutor of the project, José Luis Roda García, there emerged the possibility of working with the Canary Emergency Services (Servicio Canario de Emergencias 112) and the ICT services at the University of La Laguna.

In this sense, and thanks to the help of José Carlos González González, head of the ICT services at the University of La Laguna, we were able to establish a clear framework with some defined preconditions and restrictions, some real input data and some practical goals in the tasks which we were gradually carrying out, as we will be see below.

Having clear objectives and data which we could work with, we addressed the problem of the choice of software as a solution, a related element such as a development IDE, language maturity, methodology, hardware requirements and a large etcetera.

Finally, and in collaboration with the staff at the data centre of the School of Computer Engineers (*Escuela Técnica Superior de Ingeniería Informática*),

a multinode deployment was conducted as a final solution, together with various examples of visualisation of the generated data and proposals for the automatisation of the data analysis process in order to solve some usage difficulties.

Keywords: Big Data, Hadoop, Streaming, Python, MapReduce, Mapper, Reducer, R.

Índice General

Índice de figuras

Índice de tablas

Capítulo 1.
Justificación del proyecto

1.1 Introducción

El tema del proyecto, Big Data, se justifica en base a la situación tecnológica actual por la cual se generan datos a mayor velocidad de la que se puede procesar, y las grandes empresas se están viendo en la tesitura de eliminarlos ante la imposibilidad de su almacenamiento, perdiendo por tanto información útil después de ser procesada mediante herramientas de Big Data.

Big Data es una tecnología relativamente nueva, y en España su uso se encuentra aún en una fase inicial y poco extendida, si observamos el ***Hype Cycle* de Gartner**, el cual es un gráfico (ver Figura 1.1) que mide a las diversas tecnologías según su ciclo de vida, podemos observar la relevancia de Big Data dentro del contexto mundial actual.

Para interpretar correctamente la gráfica, simplemente poner de manifiesto que no importa que la tecnología deje de estar en su estado de "*Hype*", o sea en el punto más alto de la gráfica, sino que avance y se consolide sin desaparecer como bien está haciendo Big Data.

1.2 Desarrollo colaborativo con el STIC de La Laguna

A lo largo del desarrollo del proyecto surgió la posibilidad de trabajar de manera colaborativa con el Servicio de Tecnologías de la Información y Comunicaciones (STIC), y por ende, realizar un proyecto con datos reales, problemáticas reales y resultados igualmente verídicos.

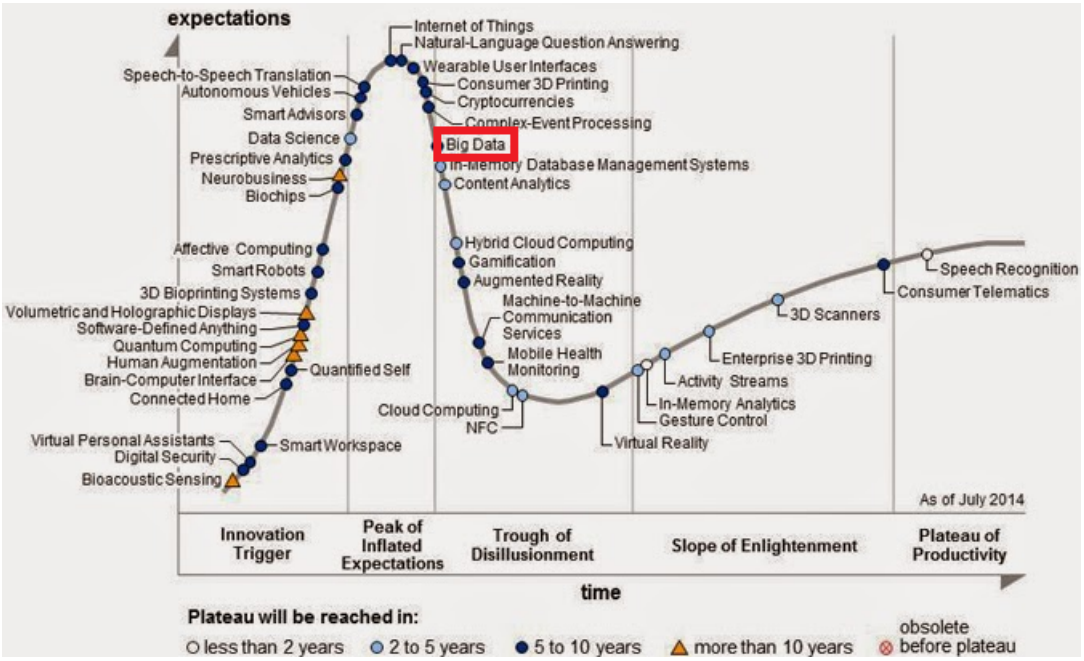

Figura 1.1. Gráfica Tecnológica de Gartner (2014).

El principal problema que se desprendió de diversas reuniones llevadas a cabo con personal del STIC fue la necesidad que tenían de poder monitorizar de manera inequívoca los accesos realizados desde un mismo dispositivo, o el desplazamiento del mismo a lo largo del campus. De esta manera se podrían inferir los patrones de uso de los usuarios a lo largo del día con el fin de reforzar la red WiFi en los puntos donde fuese necesario, siendo todo ello posible a partir de la utilización de herramientas de Big Data.

Además el proyecto serviría como un puente de cara a futuras colaboraciones entre la el Grado de Ingeniería Informática y el STIC tanto a nivel de prácticas externas como de realización de Trabajos Fin de Grado (TFG) derivados del actual o completamente nuevos.

Capítulo 2.
Estado del arte

2.1 Introducción

Con el fin de establecer un marco teórico claro sobre el que trabajar definiremos qué es Big Data, su influencia y relevancia actual así como una serie de características propias del caso de estudio que nos atañe y la resolución del mismo.

Cada día se generan en el mundo alrededor de 2.5 quintillones de bytes de datos de forma que el 90% de los datos generados en el mundo se han creado en los últimos dos años, siendo su proveniencia heterogénea. Las principales fuentes o tipo de datos que abarca Big Data son las que se ven a continuación junto con algunos ejemplos de cada uno:

1. **Datos web y de Redes Sociales**
   - Contenido web
   - Twitter
   - Facebook
   - ClickStream (datos recogidos a raíz de la navegación del usuario)
2. **Datos generados a partir de la comunicación máquina-máquina.**
   - GPS
   - RFID
   - Lectores inteligentes
3. **Grandes transacciones de datos.**
   - Gubernamentales
   - Empresariales
4. **Datos biométricos.**
   - Reconocimiento facial
   - Genéticos

5. **Datos generados directamente por los humanos.**
   - Email
   - Grabaciones de voz
   - Expedientes de todo tipo

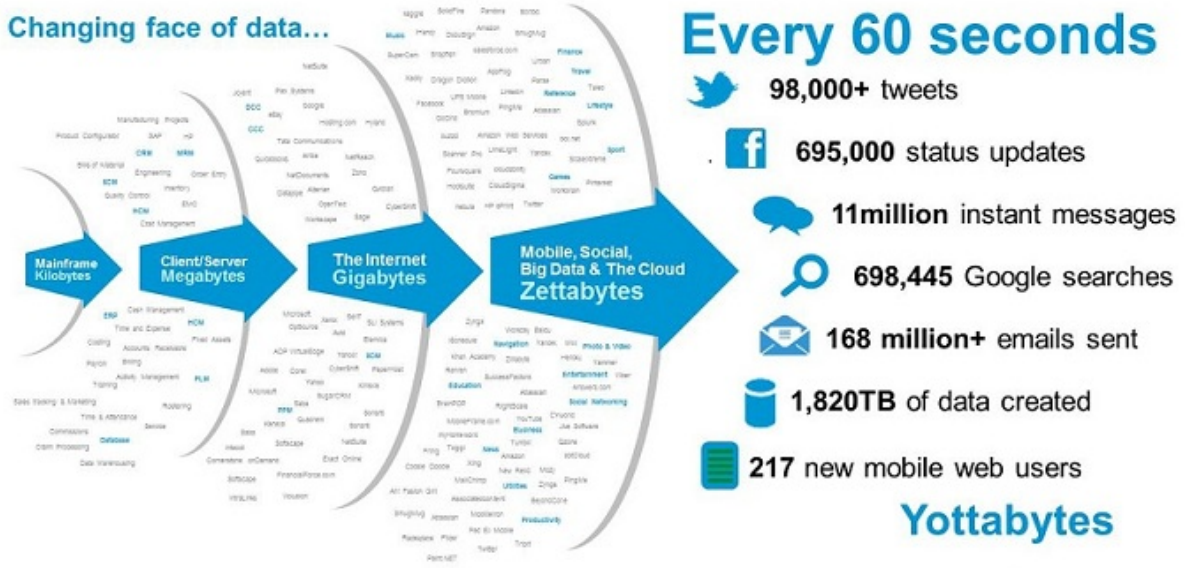

Figura 2.1. Volumen de Generación de Datos.

La definición de Big Data más ampliamente aceptada es la desarrollada por el analista industrial Doug Laney en 2001, por la cual se establecen las tres V de la Big Data:

- Volumen
- Velocidad
- Variedad

El **volumen** hace referencia a la cantidad de datos generados y la dificultad que las empresas y organizaciones encuentran a la hora de procesarlos. Las previsiones hablan de que en el año 2020, 25 mil millones de dispositivos estarán conectados a internet haciendo que los datos crezcan de manera exponencial, multiplicándose por 10 en un periodo de apenas 5 años.

Esto supone a la vez un reto y una oportunidad pero no debemos caer en la tentación de considerar el volumen de los datos como la única magnitud que defina Big Data ya que es imposible homogenizarlos para su común

tratamiento a raíz de una sola característica pues por ejemplo, como veremos a continuación en la actualidad el 90% de los datos generados son desestructurados dificultando enormemente su manejo, y por supuesto tu homogenización.

La **velocidad** a menudo se malinterpreta solo como análisis en tiempo real pero también hace referencia a la tasa de cambio en los datos, a la vinculación de datos que provienen a diferentes velocidades y también a los picos de actividad.

A su vez parece lógico pensar que la velocidad de procesamiento de los datos es un factor clave pero lo es también la velocidad de almacenamiento de los datos y su posterior acceso a los mismos. En este contexto, el almacenamiento en la nube ha revolucionado la metodología de trabajo y ha revolucionado el mundo del *Business Intelligence* permitiendo acceder a grandes volúmenes de datos de manera rápida y simultánea a partir de diferentes dispositivos generando información útil para la toma de decisiones.

La **variedad** es quizás la característica más interesante a analizar pues los datos pueden provenir de todos los ámbitos de la vida diaria como vimos anteriormente. La mayoría de estos datos ya pertenecen a las empresas pero no se hace uso de ellos, convirtiéndose en lo que Doug Laney denominó ***dark data***. Estos datos, a pesar de no ser útiles por sí mismos, después de ser analizados y procesados generan información útil de gran valor abriendo nuevas oportunidades de negocio. En función del tipo de datos podemos subdividirlos en cuatro grandes conjuntos:

- Estructurados.
- Semi-estructurados.
- Desestructurados.
- Complejamente estructurados.

En conjunto plantean uno de los grandes retos actuales como es la gestión de un volumen y una variedad ingente de datos facilitando el acceso inmediato a los mismos por parte de terceros.

Paralelamente, se suele hablar también de **variabilidad** y **complejidad** en los datos, pues estos se generan de manera inconsistente, con grandes picos provenientes de múltiples fuentes aumentando enormemente la dificultad a la hora de relacionarlos.

Un ejemplo claro de variabilidad en el flujo de datos son los trending topic en las redes sociales, que generan grandes picos de información, enfatizándose su característica variabilidad con una estructura desestructurada en la inmensa mayoría de los casos.

El Big Data puede aplicarse en campos tan diversos entre sí como la investigación médica, la seguridad, la administración pública, logística y relación con el cliente. Los expertos consideran que puede revolucionar la ciencia, la investigación, la educación, el planeamiento urbano, el transporte inteligente, el ahorro de energía, los sistemas de riesgo de análisis financiero y en definitiva cualquier sector que tengas grandes cantidades de datos susceptibles de ser analizados.

2.2 Hadoop vs Spark

Desde el punto de vista del framework utilizado para el desarrollo del proyecto desde un comienzo nos percatamos de que al estar frente a un problema multidisciplinar no íbamos a poder usar un único elemento sino una combinación de los mismos.

La primera gran decisión tuvo que ver con el *framework* de procesamiento de datos masivos a utilizar. Partiendo de la base de que debía ser software libre, las opciones fueron:

- **Hadoop MapReduce**
- **Apache Spark**

Cada uno tiene sus ventajas e inconvenientes, y la decisión fue tomada en función del momento tecnológico de ambas soluciones al comienzo del proyecto, octubre de 2014.

En ese momento, y sin haber podido todavía reunirnos con las empresas externas para analizar la problemática concreta (la primera reunión se produjo en febrero de 2015), se optó por la solución más madura y asentada, con un mayor soporte en cuanto a documentación y comunidad, a la vez que proporcionaba el mayor abanico de soluciones diferentes.

Por tanto la elección fue Hadoop MapReduce. Si comparamos algunos aspectos básicos podemos observar lo siguiente:

Rendimiento

Apache Spark en cuanto a rendimiento es por lo general más rápido que Hadoop MapReduce por el hecho principal de que procesa datos en memoria, mientras que Hadoop requiere de un acceso a disco después de cada proceso de mapeado o de reducción.

Por una parte, esta característica bajo determinadas circunstancias hace a Spark 100 veces más rápido que Hadoop; sin embargo requiere de mucha más memoria. En análisis iterativos sobre el mismo conjunto de datos, sin duda Spark es donde brilla, pero cuando los datos analizados no caben en memoria, la diferencia entre ambas soluciones se asemeja mucho, siendo esta situación el caso que nos atañe.

Facilidad de uso

En cuanto a facilidad de uso, Spark es el claro vencedor debido a las APIs con que cuenta tanto para Java, Scala y Python incluyendo también un modo interactivo por el cual se pueden ejecutar comandos con un *feedback* inmediato.

Por otra parte, Hadoop está desarrollado de manera nativa en Java y es conocida por la comunidad la dificultad que presenta a la hora de programar. Sin embargo, cuenta con un conjunto de herramientas que se incluirían dentro del llamado **ecosistema Hadoop** que simplifican notablemente esta labor como pueden ser Hive, Pig, etc.

Coste

Ambas opciones son software libre, y en cuanto a requerimientos de recursos hardware, cuentan con similares características. Sin embargo, la cantidad de memoria necesaria en Spark para procesar grandes cantidades de datos hace que esta solución cuente con un hándicap muy grande debido a las limitaciones de hardware propias del desarrollo de un Trabajo Fin de Grado.

A pesar de existir soluciones intermedias para este problema, el rendimiento de Spark se ve afectado notablemente debido a que la relación coste-espacio en disco es mucho menor que en memoria, siendo por tanto más barato ampliar disco que memoria.

Compatibilidad

En este apartado, ambas soluciones se muestran igualmente eficaces destacando ambas por su gran compatibilidad con diferentes tipos de datos y fuentes de los mismos.

Procesamiento de datos

Spark sobresale en este aspecto frente a Hadoop ya que, debido a su alto rendimiento, puede llevar a cabo tanto procesamiento en tiempo real como procesamiento por lotes, mientras que Hadoop solamente por lotes.

Tolerancia a fallos

A pesar de que ambas soluciones presenta una buena tolerancia a fallos es verdad que Hadoop MapReduce es ligeramente más tolerante debido entre otros aspectos al sistema de replicación por triplicado que implementa por defecto haciendo que la caída de un nodo, o de un conjunto de ellos no impida el bien funcionamiento de la solución.

Seguridad

A pesar de que Spark presenta autenticación por clave compartida, puede ejecutarse sobre YARN y hacer uso de HDFS accediendo por tanto a autenticación Kerberos, permisos de ficheros HDFS y encriptación entre nodos, no llega a nivel de las opciones de seguridad en Hadoop en gran medida por el volumen de proyectos existente hasta la fecha en el ecosistema Hadoop como pueden ser Know Gateway y Sentry.

Como conclusión se puede observar la diferencia entre dos tecnologías, una ligeramente más moderna como puede ser Spark frente a otra mucho más asentada y con un amplio recorrido como puede ser Hadoop. Debido al enfoque principal del proyecto y a las limitaciones del mismo se optó por la utilización de **Hadoop MapReduce frente a Apache Spark**, más concretamente YARN Hadoop siendo ésta la última versión presentando grandes mejoras de rendimiento.

Las principales razones fueron:

- Limitación de recursos hardware a nivel de memoria junto con gran volumen de datos a analizar haciendo que el rendimiento de Spark bajase considerablemente.
- El tipo de procesamiento de datos que se plantearon como hipotéticos desde un momento no sugerían (como al final ha sido) la necesidad ni de procesamiento en tiempo real ni la existencia de procesos de análisis iterativos sobre el mismo conjunto de datos, por lo que la ventaja de rendimiento de Spark quedaba minimizada y sin embargo los problemas de seguridad, la falta de documentación y el poco recorrido de la comunidad en este momento eran desventajas de demasiado peso más aún cuando se plantea una solución a un problema real de ámbito empresarial, y no algo puramente académico.

2.3 Framework

El *framework* Hadoop Mapreduce es una solución de software libre que soporta aplicaciones distribuidas siendo en la actualidad la solución más ampliamente usada por compañías de primer nivel como Yahoo, Ebay, Facebook, IBM, etc,, ya que permite trabajar con miles de nodos y con PetaBytes de datos en parte gracias a la actualización de Hadoop a su versión 2.0.

Se suele hablar de Ecosistema Hadoop debido a la cantidad de proyectos, servicios, librerías, APIs, etc., que aglutina pudiendo inferir que está compuesto por tres elementos base:

- **HDFS**: básicamente un sistema de ficheros distribuido, escalable y portátil.
- **MapReduce**: un JobTracker al cual las aplicaciones clientes envían trabajos.
- Una serie de **aplicaciones paralelas** que enriquecen el ecosistema Hadoop, tales como Pig, HBase, Hive, Hue, etc.

Sin embargo, para el desarrollo de este proyecto utilizaremos Hadoop 2.0 el cual incorpora un cuarto elemento, **YARN,** liberando al motor MapReduce

de las tareas de gestión de los recursos del clúster, agilizando el procesamiento de los datos y la ejecución de las tareas.

También incorpora otro elemento, Tez, que sirve como puente entre el procesamiento por lotes más tradicional de Hadoop y el interactivo más propio de competidores como Apache Spark. Permite a su vez personalizar la arquitectura de ejecución con el fin de procesar cálculos complejos de manera más eficaz potenciando el *framework* en aquellas situaciones donde Spark destacaba por su rendimiento en casos como las ejecuciones en tiempo real, o cercano al real.

De esta manera la arquitectura de Hadoop 1.0 pasaría a ser la siguiente en Hadoop 2.0:

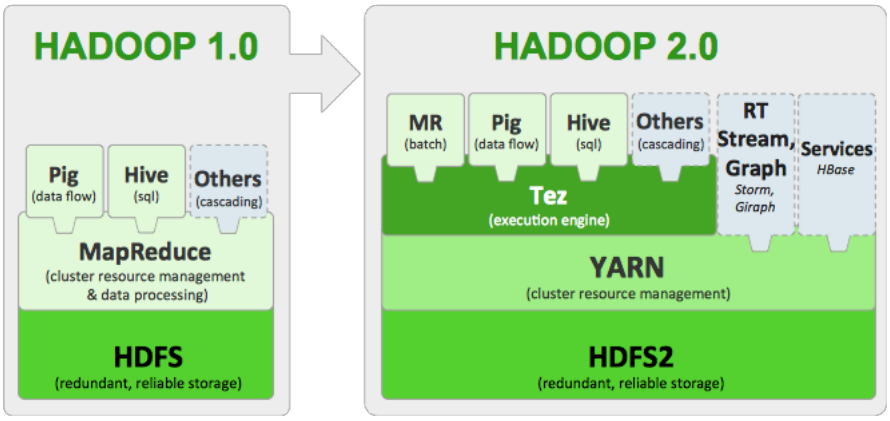

Figura 2.2. Actualización de la arquitectura de Hadoop.

A su vez cabe destacar que el sistema HDFS, está especialmente diseñado para gestionar archivos de gran tamaño. Se encarga de trocear y distribuir el contenido de grandes ficheros de datos en bloques o porciones generalmente de 64 MB a través de múltiples máquinas. Su fiabilidad se fundamenta en el replicado de datos a través de diferentes hosts, replicándose por defecto en 3 host cada bloque de datos. De esta manera, los host se pueden comunicar entre ellos y reequilibrar el flujo de datos, mover los datos y conservar una alta replicación en los mismos a fin de garantizar el correcto funcionamiento de la aplicación.

En cuanto al esquema de funcionamiento interno de Hadoop MapReduce podemos descomponerlo de la siguiente manera:

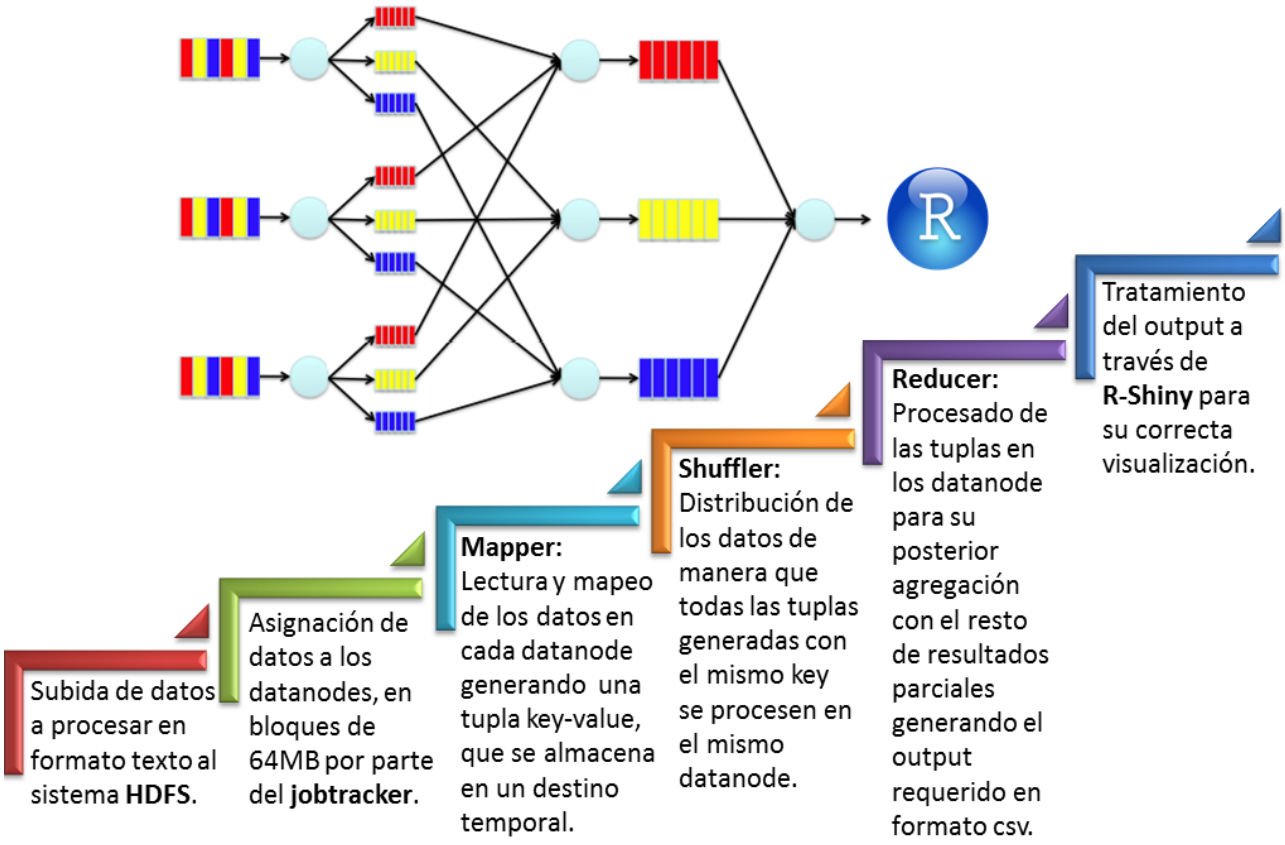

Figura 2.3. Workflow interno – Hadoop MapReducer

Como queda ejemplificado en la Figura 2.3 anterior, es un proceso secuencial por el cual a través de una serie de datos de gran volumen almacenados en el sistema HDFS, éstos se particionan en bloques más pequeños de 64MB por parte del JobTracker en el nodo padre o NameNode. Estos se distribuyen a los nodos hijos o DataNodes para llevar a cabo la lectura y mapeo de los mismos generando como salida una tupla *key-value* que se almacena en un destino temporal. En este punto, y gracias al servicio Shuffler, se ordenan estas tuplas por la *key* de manera que se garantice el procesamiento de todos los elementos con un mismo *key* en un mismo nodo. El último paso es el procesado de estas tuplas en los DataNodes y su posterior agregación con el resto de resultados parciales hasta generar resultado en el formato deseado.

Si bien con esto finaliza el proceso MapReduce, se suele añadir un último paso que es la visualización de los datos obtenidos, como por ejemplo, con el software estadístico R y framework R-Shiny.

Capítulo 3. Problemática

A la hora de enfocar el proyecto se optó por darle una vertiente eminentemente práctica y realizarlo en colaboración con empresas o entidades, las cuales serían las encargadas de proveer de datos sobre los que trabajar así como establecer las tareas a desarrollar.

En este contexto surgió la posibilidad de trabajar en colaboración con el **STIC** de la Universidad de la Laguna el cual es el encargado del mantenimiento y gestión de una red compuesta por más de 100 servicios telemáticos accesibles desde 26 edificios, y compuesta por más de 100 dispositivos de red entre WiFi y cableados.

El STIC contaba con una aplicación desarrollada con la herramienta de Business Intelligence Pentaho Community la cual pretendía analizar las grandes cantidades de logs generados a raíz de la gestión de la red de comunicación de la universidad con resultados muy discretos y específicos.

El objetivo de esta herramienta era combinar la información obtenida de diversas fuentes, principalmente log de acceso Apache Log, fichero log de asignación DHCP, y los archivos log generados por los controladores de los diferentes dispositivos WiFi, con el fin de poder determinar el número y tipo de accesos por edificio para su análisis.

Si bien la herramienta hacía esto de una manera aproximada y sin la posibilidad de aplicar filtros, el procesado de la franja horaria de menor uso de la red de un solo día suponía un **proceso de 3 días de ejecución** por lo que quedó probada la posibilidad de ejecución de la tarea, pero no así la viabilidad de la misma.

De esta manera se planteó la posibilidad de llevar a cabo un proceso por el cual, a raíz de computación distribuida y con herramientas específicas de Big Data, este tiempo se viene reducido notablemente.

3.1 Restricciones

A raíz de una serie de reuniones con José Carlos González González (Jefe de Servicio del STIC) se planteó la problemática total sobre la que trabajaríamos y se establecieron una serie de restricciones que en gran medida crearon la estructura de este proyecto.

Entre ellas destaca la **utilización de Python** como lenguaje de desarrollo para su fácil mantenimiento por parte del personal del STIC, ya que es el lenguaje mayoritario con el que trabajan.

A la hora de trabajar con Hadoop esto supone un factor clave puesto que de manera nativa Hadoop está desarrollado en Java, y la inmensa mayoría de documentación, así como librerías y servicios asociados dentro del ecosistema Hadoop están desarrollados en dicho sentido.

Frente a esta situación y siempre primando el rendimiento se analizaron las diferentes alternativas y se compararon a fin de garantizar el mejor de los resultados en aspectos tan diversos como velocidad de lectura, tiempo de procesamiento, velocidad de escritura, etc.

De esta manera podemos observar lo siguiente:

	Java	Streaming*	mrjob*	dumbo*	hadoopy*
FILE: bytes read	22,726,677,381	0.94	1.34	2.55	1.97
FILE: bytes written	33,468,535,411	0.93	1.35	2.57	1.99
HDFS: bytes read	21,934,848,598	1.00	1.00	1.00	1.00
HDFS: bytes written	7,629,045,090	1.00	0.99	1.00	1.06
Map output bytes	12,978,686,993	0.91	1.40	2.11	2.11
Reduce shuffle bytes	11,336,515,993	0.92	1.35	2.53	1.97
Reduce input records	428,755,439	1.04	1.00	1.46	1.04
Time spent all maps (ms)	14,256,288	1.37	5.98	2.39	3.76
Time spent in all reduces (ms)	4,348,716	1.76	8.91	6.14	4.86
CPU time (ms)	14,016,540	1.17	4.68	3.68	2.76
Job run time (s)	1,074	1.54	7.31	3.90	4.20

*Ratios are relative to Java values

Figura 3.1. Opciones de desarrollo en Python para Hadoop.

A partir del análisis de los datos obtenido se concluyó que la mejor opción era desarrollar a través de la librería **Streaming**, la cual presenta una radios de rendimiento semejantes a Java, e incluso mejorando sus prestaciones en algunas ocasiones como se puede observar en la Figura 3.1 anterior.

También era importante constatar que Streaming tuviese un abanico de funcionalidades semejante a Java para el desarrollo de una solución integral a

un problema de Big Data como el que teníamos entre manos por lo que se analizó en detalle concluyendo que incluía las funcionalidades básicas, soportaba el formato de datos con que íbamos a trabajar y la documentación si bien no era la mejor si ofrecía unas garantías suficientes como se puede observar en la Figura 3.2.

	Java	Streaming	mrjob	dumbo	hadoopy	pydoop
Underlying framework	Hadoop	Hadoop Streaming	Hadoop Streaming	Hadoop Streaming	Hadoop Streaming	Hadoop Pipes
Ease of installing framework	Easy with CDH (Whirr for cloud)	Easy with CDH (Whirr for cloud)	pip on client only	Build as egg, manually distribute to cluster	pip on client only (performance penalty) or manually install across cluster	Failed to build
Documentation quality	Extensive/complex	Good	Very good	Poor	Good	Good
Work with non-text objects	Yes	Manual ser/de	Built-in JSON	Built-in typedbytes	Built-in typedbytes	Unclear
Data formats supported	All	Text/manual	Text, Repr (string), JSON, Pickle	Text, SequenceFile (typedbytes), any Java InputFormat	Text, SequenceFile (typedbytes), Pickle	Text, SequenceFile for I/O (but unclear)
Implement custom SerDe	Yes	manual	Yes	Yes	No	No
Integrate with arbitrary Java classes	Yes	Yes	Experimental	Yes	Unclear	Unclear
Multistep MapReduce workflows	Yes, but awkward	No	Yes	Yes	No	No
Implement Partitioner in Python		No	No	No	No	Yes
HBase integration	Yes	No	No	Experimental / unofficial	Experimental	No
Support for AWS/EMR	Manual	Yes; EMR exposes Streaming API	Yes; flawlessly integrated through boto	Manually setup EC2 cluster	Setup EC2 with supplied Whirr config	No
Actively developed			Very	Somewhat	Yes	Yes
Commits in last year			1063	19	167	272
SLOC add+delete in last year			66094	1314	75091	172223
First commit			10/13/2010	6/15/2008	10/17/2009	3/10/2009
Sponsored?			Yelp	Individual (Last.fm?)	Individual	CRS4
Hosted on			GitHub	GitHub	GitHub	Sourceforge
License			Apache v2.0	Apache v2.0	GPL v3	Apache v2.0

Figura 3.2. Funcionalidades de Python frente a Java.

La última restricción venía impuesta con el *framework* en sí, ya que los procesos pueden ser abordados desde el *framework* MapReduce. Concretamente, son abordables sólo aquellos que se pueden disgregar en las operaciones de **map()** y de **reduce()**. Las funciones Map y Reduce están definidas ambas con respecto a datos estructurados en tuplas del tipo (clave, valor), característica que tuvimos presente a la hora de abordar la resolución de las tareas.

3.2 Metodología de trabajo

Uno de los mayores retos cuando se trabaja con grandes cantidades de datos es verificar la exactitud de los resultados obtenidos, siendo éste un aspecto crítico para el personal del STIC.

De esta manera se diseñó una metodología de trabajo por la cual se llevaría a cabo un desarrollo progresivo de la solución que permitiesen ir controlando los resultados a raíz de un conjunto de datos de entrada suficientemente

grande como para considerarse Big Data, pero a su vez perfectamente acotables para su análisis.

De esta manera, el desarrollo se fragmento en hitos, cada uno de los cuales con una tarea definida, perfectamente documentada, y con una serie de archivos de datos como input para analizar.

Así, el desarrollo fue dirigido en gran medida por el STIC, ya que eran los encargados de validar los resultados obtenidos y proponer la siguiente tarea a realizar. Haciendo un símil y salvando las distancia, la metodología del desarrollo llevado a cabo se asemeja en gran medida a la metodología Scrum, siendo esta un proceso iterativo con reuniones de puesta en común donde el Scrum Master (en este caso el STIC) lleva a cabo la gestión de cada iteración.

La plataforma usada para la gestión del proyecto por parte del STIC fue Google Drive haciendo uso de la siguiente estructura:

- Tarea con número de referencia y título de la misma para su fácil seguimiento.
  - Directorio de inputs con el conjunto de datos reales aportados por el STIC.
  - Directorio de documentación de la tarea a realizar creado por el STIC y puesto en común con el tutor del proyecto.
  - Directorio de outputs generados por el alumno con la solución en el formato requerido donde si es necesario se adjuntan anotaciones, dudas y comentarios.

De esta manera se concluyó que el plan de trabajo quedaría de la siguiente manera:

- **Recopilar información sobre análisis de log**.

- **Instalación y desarrollo de tareas básicas** de Hadoop básico tanto en Java como en Python como herramienta de acercamiento al tema.

- Descargar distribuciones de Hadoop (Cloudera, Hortonworks, MapR), con el fin de ir probando sus características para elegir la que mejor encaja en el proyecto a desarrollar si procediese.

- **V0.5 Análisis de una carpeta de ficheros de log**.
  El sistema en un sólo nodo computará los datos contenidos en un directorio, recorrerá todos los ficheros de log que existan en ella y como salida generará un fichero en formato CSV con los nombres de los ficheros encontrados y el número de líneas de log detectado en cada uno.

- **V1.0 Análisis de un fichero de log sencillo un sólo nodo de análisis.**
  Para un sólo fichero de log de un sólo servidor de la franja de servidores del Campus Virtual, obtener el número de accesos agrupados por hora.

- **V1.5 Análisis de un fichero de log sencillo tres nodos de análisis.**
  Para un sólo fichero de log de un sólo servidor de la granja de servidores del Campus Virtual, obtener el número de accesos agrupados por hora repartiendo el cálculo en tres nodos de análisis.

- **V2.0 Análisis de varios ficheros de log sencillo de un sólo día en varios nodos de análisis.**
  Para el conjunto de ficheros de log de un sólo día de varios servidores del Campus Virtual, obtener el número de accesos agrupados por hora globales sumando los accesos a cada servidor y para cada franja horaria.

- **V2.5 Análisis de varios ficheros de log sencillo de un sólo día en varios nodos de análisis con restricciones de día y hora.**
  Para el conjunto de ficheros de log de un sólo día de varios servidores del Campus Virtual, obtener el número de accesos agrupados por hora globales sumando los accesos a cada servidor y para cada franja horaria filtrando por las restricciones de día desde, hora desde y día hasta y hora hasta que se especifiquen en un fichero de filtro de entrada al sistema.

- **V3.0 Análisis de los log de DHCP.**
  Se hace para determinar las MAC de los dispositivos que se conectan a la red en cada instante. El sistema leerá los ficheros de log de los servidores DHCP y como salida generará un nuevo fichero con las direcciones MAC asociadas a cada dirección IP en cada momento del tiempo.

- **V4.0 Enlazando datos no estructurados, correlacionando elementos en distintos instantes del tiempo.**
  Para una línea de log de acceso al Campus Virtual, el sistema tendrá que determinar cuál es la dirección MAC asociada a la dirección IP que hizo el acceso. Como salida ya se podrá tener la tupla de Fecha, Hora, IP, MAC y Objeto de acceso.

- **V4.5 Desplegando 4.0 para un periodo de log especificado en los filtros de entrada.**
  En este apartado hay que enlazar todo lo anterior y obtener la tupla del 4.0 para todos los log de un periodo determinado en el filtro de entrada.

- **V5.0 Analizando las conexiones WiFi.**
  Analizar los ficheros de log de las diferentes controladoras WiFi de la ULL para obtener las tuplas Fecha, hora, MAC, Punto de Acceso WiFi.

- **V5.5 Analizando las conexiones WiFi y geoposicionando los accesos.**
  Analizar los ficheros geolocalizados de los puntos de acceso WiFi y enlazar al apartado anterior las coordenadas de posicionamiento del acceso detectado.

- **V6.0 Enlazando todo.**
  Enlazar todos los apartados anteriores para obtener la tupla Fecha, Hora, Acceso Campus Virtual, IP, MAC, Punto de Acceso, y

Coordenadas GPS, para un periodo especificado en el filtro de entrada.

Capítulo 4.
Fases y Desarrollo del proyecto

4.1 Instalación de Hadoop 2.0

El primer paso en todo desarrollo es la instalación de las dependencias y requerimientos para llevar a cabo el proyecto.

En este caso debido a las limitaciones de hardware con que contaba, se optó por la instalación de un clúster mononode de Hadoop para el desarrollo de las tareas a realizar, utilizando como sistema operativo GNU Linux Ubuntu, en su versión 14.04 LTS con el fin de asegurarnos un soporte constante a lo largo de todo el proceso de desarrollo, y entre sus requisitos destaca la necesidad de utilizar Java 1.5 o superior, siendo recomendado instalar Java 1.7.

Otra de las recomendaciones es utilizar un usuario dedicado a la ejecución de Hadoop, por lo que se generó un usuario llamado hduser dentro del grupo hadoop con el fin de separar ésta instalación del resto del software instalado en el equipo.

A continuación, Hadoop requiere de acceso SSH para administrar los nodos, la máquina local, etc., pero en nuestro caso para tener acceso a localhost, por lo que generamos una clave SSH y la añadimos al conjunto de claves autorizadas del usuario creado anteriormente.

Una vez hecho lo anterior, descargamos Hadoop y lo descomprimimos donde queramos, asignando como dueño al usuario hduser, y como grupo hadoop.

Modificamos el fichero .bashrc propio del usuario hduser de manera que queda configurado, estableciendo las variables de entorno propias de Hadoop, la versión de java a utilizar así como un conjunto de alias para atajos futuros entre otras cosas.

El siguiente paso es desactivar IPv6 ya que Ubuntu usa la dirección 0.0.0.0 para diferentes propósitos de lo que configuraremos a continuación para Hadoop.

A la hora de configurar el sistema de ficheros HDFS se comienzan a modificar los archivos de configuración propios de Hadoop, siendo éste uno de los aspectos críticos en el proceso de instalación ya que modificaremos lo siguiente ficheros:

- yarn-site.xml
- core-site.xml
- mapred-site.xml
- hdfs-site.xml

Modificamos el fichero yarn.site.xml con el fin de configurar desde el servicio shuffle (básico para el correcto funcionamiento de Hadoop) hasta aspectos tan concretos como la asignación de memoria a cada uno de los nodos, el número de nodos virtuales que crearemos localmente, la cantidad de memoria física asignada a cada nodo, la cantidad de memoria de cada una de las particiones en las que se descompone el fichero de datos inicial y un largo etcétera. Esta tarea es básica, y depende completamente de los recursos hardware de que se dispongan. La correcta asignación de recursos es crucial para mejorar el rendimiento de la solución, no existiendo una configuración maestra estandarizada lo que implica un proceso de prueba y error.

Como su propio nombre indica, a través del fichero core-site.xml configuramos parámetros a nivel de core que sustituyen la configuración por defecto con que cuenta Hadoop.

Mediante el fichero de configuración mapred-site.xml establecemos que el servicio yarn se encargará de la gestión de los recursos del clúster liberando por tanto el servicio MapReduce de esta tarea y por ende agilizando el proceso.

A través del fichero hdfs-sites.xml establecemos la configuración básica del sistema HDFS, estableciendo entre otros parámetros la ubicación de los directorios que serán usados tanto para el namenode como para el datanode de manera local.

El siguiente paso es crear los directorios que servirán como datenode y namenode de manera que coincidan con lo establecido con lo en el fichero de configuración hdfs-sites.xml, y le damos formato. Es importante destacar que cada vez que se formatea el sistema HDFS se pierde la información contenida en el por lo que exceptuando casos puntuales solo se llevará a cabo esta operación en el momento de la instalación.

El último paso es lanzar los demonios propios de Hadoop comprobando el correcto funcionamiento de cada uno de los elementos. Con este demonio se inician los siguientes elementos básicos:

- NameNode
- DataNode
- JobTracker
- TaskTracker

Una manera sencilla de comprobar si los servicios están levantados es escribir jps en la terminal de Linux y verificar que efectivamente están funcionando correctamente.

Uno vez hecho esto, ya tendríamos Hadoop preparado para ejecutar tareas, contando además con una interfaz de seguimiento de las mismas a nivel web cuya ruta dependerá de la configuración establecida en los ficheros anteriormente nombrados.

Cabe destacar además que la librería Streaming forma parte de la instalación básica de Hadoop por lo que no requiere configuración o instalación extra exceptuando el tener instalado Python en el ordenador.

4.2 Elección de IDE

Una vez instalado y configurado el framework se comienza con el desarrollo progresivo del proyecto, cobrando especial importancia el IDE de desarrollo a utilizar.

Entre las diferentes alternativas para Linux destaca una sobre el resto, **PyCharm**.

PyCharm es un IDE creado por la compañía JetBrains especializada en software de aumento de productividad, como por ejemplo ReShaper para Visual Studio.

Este enfoque era básico debido a mi nulo conocimiento previo sobre Python y las siguientes características fueron claves en el desarrollo del proyecto:

- Finalización de código.
- Formateo automático de código.
- Refactorización.
- Importación automática.
- Navegación de código con un solo clic.
- Soporte de entornos virtuales e intérpretes de Python 2.x, 3.x, Pypy, Iron Python y Jython.

4.3 Ejecución de soluciones con Python

La estructura de trabajo de Hadoop es por lo general sencilla y se basa a grandes rasgos en un proceso de filtro y adquisición de datos de entrada Mapper, y un proceso posterior de agregación o análisis Reducer.

De esta manera la metodología con la que se abordaba una nueva tarea siempre fue bajo este prisma generando en el proceso un conjunto de scripts:

- Mapper.py
- Reducer.py

A esto hay que añadir el conjunto de datos de entrada que normalmente se encuentran almacenados en el HDFS, y el consiguiente output que se almacenaba igualmente en el HDFS.

De ésta manera la morfología de la sentencia de ejecución haciendo uso de la librería Streaming sería la siguiente:

bin/hadoop jar ruta a la ubicación de la librería Streaming dentro de Hadoop.

-files ruta al conjunto de ficheros necesarios para la ejecución, como el mapper.py y el reducer.py, filtros externos que queramos configurar, etc.

-mapper nombre del fichero incluido en la opción "files" que implementa el proceso de mapeo de los datos, normalmente mapper.py

-reducer nombre del fichero que implementa el proceso de reducción de los datos, normalmente reducer.py

-input conjunto de archivos con un gran volumen de datos de entrada se encuentran almacenados en el HDFS.

-output ubicación y nombre del fichero con los resultados obtenidos dentro del sistema HDFS.

4.4 Workflow de ejecución interna de Hadoop

A nivel general la estructura de los scripts generados responden al esquema de la siguiente Figura 4.1 en la que se muestra un pseudocódigo del flujo de ejecución interna de una tarea MapReduce.

```
#mapper
def read_input(file):
    for each line in stdin:
        yield (line + name_of_file)

def main():
    data = read_input(sys.stdin) #generator
    for each line in data:
        result = line.match(regex_expresion)
        stdout = (result, 1) #key-value output


#reducer
def main():
    data = sys.stdin

    for dif_ip and dif_path in data:
        try:
            dictionary = line.values
        exception:
            dictionary(dif_ip, dif_path) += 1

    stdout "Cabecera"
    for x in sorted(dictionary):
        stdout (key, sum_values)
```

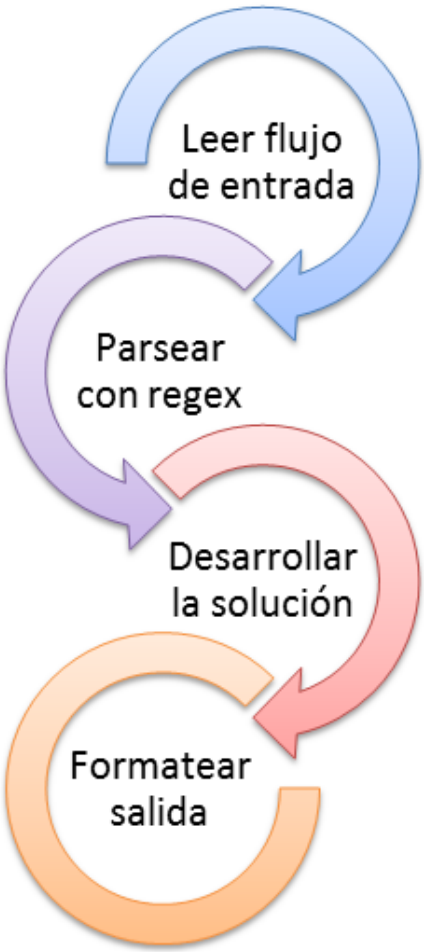

Figura 4.1. Workflow MapReduce – Pseudocódigo.

A esta estructura básica se le añaden las particularidades de cada tarea en función muchas veces de la aplicación de filtros externos al propio input general de datos que modifican el flujo de trabajo.

4.5 Conjunto de tareas a desarrollar

Como extensión de lo explicado en el apartado 3.2 me gustaría destacar una serie de hitos que supusieron un reto a nivel de código o que ilustran perfectamente el desarrollo progresivo que se llevó a cabo y como a través de él en cierta medida se puede comprobar la verosimilitud de los resultados obtenidos.

Durante el desarrollo del punto V0.5 por el cual se pretendía realizar un conteo simple del número de registros por cada fichero de acceso, se aprendió que paralelamente a la ejecución de cada job, Hadoop genera un conjunto de **variables de entorno** de uso propio con información muy útil para el desarrollador. Por ejemplo, en este caso la variable de entorno que nos interesa es 'mapreduce_map_input_file' que como su nombre indica contiene la ruta dentro del sistema HDFS al archivo desde el que se está leyendo a través del flujo de entrada stdin. De esta manera podemos fácilmente relacionar cada entrada con su fichero de origen a través de una estructura (key, value) que posteriormente será reducida.

Cabe destacar una diferencia clave en el input que recibe el proceso reducer generado mediante tareas realizados con Hadoop nativo, o sea con Java, y el generado con Streaming. De manera nativa el reducer recibe los datos siguiendo la estructura **reduce(k, Iterator[V])**, mientras que Streaming genera una línea por valor en la que se incluye la clave, de manera que se debe realizar un pequeño parseo sobre el flujo de entrada stdin que reciba reducer.

A raíz de la resolución de la tarea V2.0, la estructura base del código quedó definida. Primeramente, se actualiza la forma de procesar los datos y se introducen **iteradores y generadores** con el fin de mejorar el rendimiento. Por otra parte, en ella se realiza una agrupación de datos provenientes de múltiples servidores en función de algún parámetro, en este caso la hora, a la

vez que se discrimina en función de otros elementos, en este caso franja horaria y servidor de origen.

En este punto empieza a cobrar especial relevancia el trabajo con **expresiones regulares** y se obtiene la estructura base con la que parsear los log de acceso.

En el desarrollo de la tarea V4.0 introducimos el análisis de DHCP como parte del desarrollo. Normalmente Hadoop se utiliza para, a partir de unos datos de entrada masivos, generar un resultado de mucho menor tamaño, pero esta tarea va en contra de ese principio. Por esta razón la resolución de esta tarea se realizó mediante la **concatenación de dos jobs independientes**. El primero de ellos se encarga de filtrar el contenido del log DHCP de acuerdo las los requisitos del STIC y obtener una serie de datos, como puede ser la MAC asociada a cada IP en un momento determinado de tiempo. El siguiente paso es cruzar estos resultados con los log de acceso y de esta manera establecer la MAC asociada a cada uno de los registros de acceso.

Finalmente, mediante la resolución de la tarea V6.0 introducimos la **incorporación de librerías externas**, en este caso de Python, al ecosistema Hadoop. De esta manera se añade toda la potencia y funcionalidad de Python al análisis de datos de Hadoop, utilizándose en este caso la librería netaddr con el fin de determinar si una IP pertenece o no a un rango de IPs, y de esta manera, georreferenciar los datos.

4.6 Estructura de los datos de entrada

Como se ha comentado anteriormente, una de las principales características es el trabajo con datos reales suministrados por el STIC, con el fin de satisfacer una problemática particular. La principal fuente de datos para el proyecto son los archivos de log de acceso que se generan de manera automática mediante el uso de la red de la ULL por parte de todos los usuarios y los dispositivos que conectados a la misma. En la actualidad, la cantidad de datos que se generan diariamente como consecuencia del monitoreo de la red excede los 10 GB por hora, haciendo que realmente estemos ante un problema de Big Data.

Esto es debido sobre todo a lo extensa de la red, dando acceso a más de 10.000 dispositivos, y servicio a más de 26 edificios, que suman entre ellos más de 1.000 dispositivos de red, entre WiFi y cableados interconectando a los usuarios con los servicios.

Por tanto, la principal fuente de información con la que trabajar son los archivos de acceso, destacando que dichos log carecen de información sensible y por tanto no necesitan de su anonimización. Estos presentan una morfología característica siendo una concatenación de un **log de Apache** normal, pero desordenado, junto con una serie de datos que añade el sistema gestor de log propio de la ULL. Los campos se encuentran separados por espacios, representando el símbolo "-" la ausencia de esos datos.

Podríamos clasificarlos como **datos semi-estructurados**. De esta manera podemos observar lo siguiente:

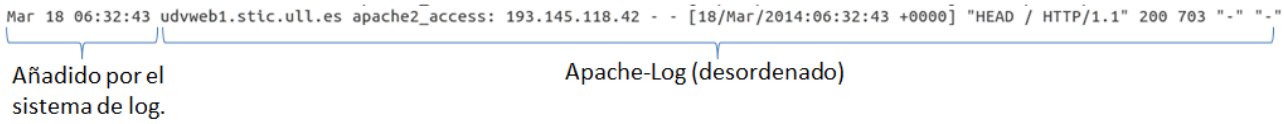

Figura 4.2. Fragmento de log de acceso del STIC.

A parte de codificar la hora del acceso o petición, podemos observar el tipo de petición que se está llevando a cabo, el HTTP Status de la petición, el tamaño de la misma en bytes, etc., si bien es cierto que se puede observar la discrepancia con los log de Apache comunes. Esto implica el desarrollo de expresiones regulares específicas para el caso de estudio.

Otras de las fuentes de datos del proyecto fueron los log del tipo DHCP. Su morfología es completamente diferente a los log de Apache, como se puede observar:

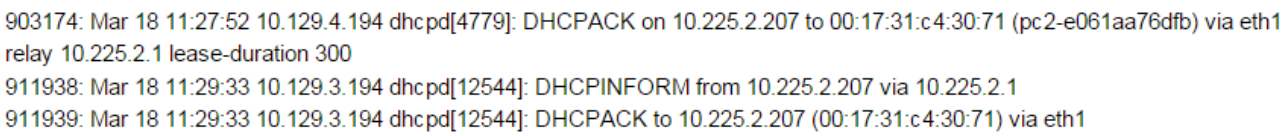

Figura 4.3. Fragmento de log DHCP.

Un aspecto relevante a la hora de hablar de los datos de entrada es poner de manifiesto la versatilidad de Hadoop con respecto al formato de los datos de entrada permitiendo que éstos se encuentren comprimidos en los formatos más comunes como zip, rar y tar.

Generalmente los ficheros de datos en este proyecto abarcan un periodo de un día completo para un servidor en específico. De esta manera el peso medio de cada archivo es de 170 MB incluyendo aproximadamente 500.000 registros por fichero. El análisis de un periodo extenso como puede ser un mes de todos los servidores supone un volumen de datos a analizar de varios cientos de GB por lo que se contempló la posibilidad de utilizar los datos comprimidos.

Una vez analizados los resultados, se constató el empeoramiento del rendimiento debido a dos factores claves:

- La replicación de los archivos para garantizar la tolerancia a fallos, así como el movimiento de los mismos para balancear la carga de los nodos, hacía mucho más lento el proceso al circular por la red bloques de 170 MB en lugar de 64 MB.
- Imposibilidad de dividir estos ficheros en bloques de 64 MB óptimos para Hadoop, por lo que en caso de que un log de un servidor pesase 250 MB y otro simplemente 100 MB, se debería esperar a la ejecución del mayor para continuar haciendo que no se aprovechasen al máximo los recursos hardware disponibles.

4.7 Regex

Una de las ideas básicas de Hadoop, como bien se expuso anteriormente, es "*guardar una vez, y leer múltiples veces*", ya que el proceso de enviar y almacenar información entre los diferentes nodos es donde mayor tiempo se emplea y por consiguiente donde disminuye el rendimiento de la solución desarrollado. Debido a ello, se optó desde un comienzo por la utilización de expresiones regular para parsear el flujo de entrada proveniente de los ficheros de datos con la intensión de que el tráfico de datos en la red fuese el menor posible descartando toda la información no relevante para la correcta resolución de la tarea.

De esta forma, se hizo uso de un depurador web libre de expresiones regulares, regex101, debido a las funcionalidades que incorpora, como explicaciones en tiempo real, detección de errores y resaltado automático.

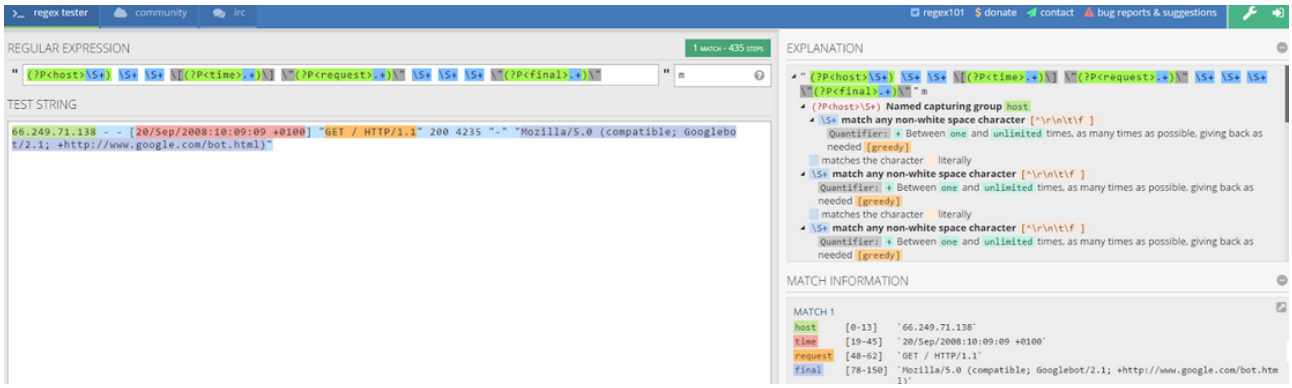

Figura 4.4. Caso de uso de Regex101.

Como se puede observar, ofrece una representación visual de las coincidencias de la cadena a analizar con la expresión regular desarrollada e incorpora ayuda en línea para facilitar el proceso de creación de expresiones regulares válidas.

4.8 Utilización de librerías Python

Durante el desarrollo del proyecto se puso de manifiesto que, si bien la librería Streaming incluía la mayoría de funcionalidades propias de Hadoop, no podía compararse con la cantidad de servicios, librerías y APIs desarrolladas específicamente para la versión nativa de Hadoop, o sea Java.

Sin embargo Streaming permite sin demasiados cambios integrar toda la potencia de las librerías actualmente generadas para Python, independientemente de que no fuesen diseñadas para tareas de Big Data, dentro de la ejecución propia de la solución.

De esta manera sencilla se puede acceder a toda la potencia de Python de la siguiente manera:

- Descargar la librería en cuestión.
- Comprimir en formato zip.
- Renombrar con extensión .mod.

- Incluir en la sentencia de ejecución como un fichero externo más dentro de la opción –file.
- Modificar el mapper.py de la siguiente manera para que incluya las funcionalidades propias de la nueva librería:

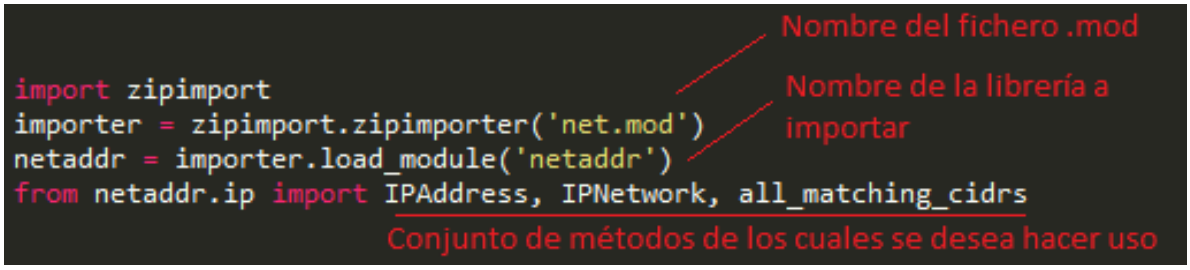

Figura 4.5. Modificaciones sobre el mapper.py

De manera análoga ocurre con librerías de Python diseñadas por nosotros, haciendo que las posibilidades en este sentido sean infinitas con el fin de ayuda a no duplicar código y a la limpieza del mismo a la vez que dotamos a la aplicación de funcionalidad extra.

4.9 Consideraciones a la hora de programar

Unos de los aspectos más relevantes es la optimización del código generado con el fin de maximizar el rendimiento de la solución, más incluso cuando hablamos de Big Data, donde una mala configuración se repite millones de veces ralentizando el proceso enormemente.

Bajo esta premisa el desarrollo de las tareas propuestas por el STIC se han llevado a cabo priorizando el tiempo de ejecución final frente a la legibilidad del código desarrollado.

Por poner algunos ejemplos, al comienzo del proyecto el desarrollo se realizó de manera más intuitiva haciendo uso directo del flujo de entrada stdin propio de Hadoop, y se analizaba directamente este flujo. Con la evolución del proyecto esto se modificó y se comenzó a hacer uso de iteradores y generadores a la hora de trabajar con los datos.

La principal ventaja de esta técnica es que los elementos de una secuencia no son generados hasta que realmente se necesita reduciendo notablemente el consumo de memoria y el gasto computacional.

Lo mismo sucede con las estructuras usadas para almacenar las grandes cantidades de datos generados durante el proceso de análisis. Si analizamos por ejemplo la complejidad media y máxima de una serie de operaciones como pueden ser el insertado o la ordenación en función de la estructura de datos utilizada, se observan grandes diferencias que repercutirán en el rendimiento de la aplicación. De esta manera, si nos interesa que los datos almacenados se encuentren ordenados para su uso futuro, conviene hacer uso de un diccionario en lugar de una lista, por ejemplo, donde el orden de complejidad tanto medio como máximo es de $O(n \log n)$ mientras que el diccionario por su propia morfología ya los inserta ordenador con una complejidad media de $O(1)$.

El código desarrollado también es optimizable a nivel de operaciones básicas, ya que por ejemplo "$x < y < z$" es más rápido que "$x < y$ and $y < z$" a pesar de que ambas operaciones son equivalentes.

Este tipo de consideraciones a la hora de desarrollar son vitales puesto que pueden mejorar el rendimiento en porcentajes altísimos simplemente modificando la estructura y los operadores utilizados, no así en este caso la lógica.

4.10 Despliegue en clúster multinode

A pesar de que el grueso del desarrollo se llevó a cabo en un clúster mononode donde solo se podía simular un sistema distribuido mediante la virtualización de los nodos, la potencia de Hadoop radica en la computación distribuida a través de un sistema de clustering.

De esta manera surgió la posibilidad en colaboración con el Centro de Cálculo de la ETSII de desplegar un pequeño clúster de entre 5 y 7 nodos para llevar a cabo la ejecución de tareas de Big Data relacionadas con el proyecto.

La primera idea fue la utilización para el despliegue de una de las múltiples Sandbox con que cuenta el ecosistema Hadoop tales como Cloudera, HortonWorks o MapR. La principal ventaja de realizar el despliegue de esta manera era reducción en cuanto a la complejidad de la configuración que se realizaba de manera transparente para el usuario. A su vez estas sandbox

integraban un conjunto de servicios tales como Pig, Hive, Hue, HBase, etc., y contaban con una extensa documentación así como un gestor de workflows muy intuitivo para facilitar el uso por parte del STIC de las soluciones desarrolladas.

Sin embargo esto no fue posible debido a la gran cantidad de requisitos hardware que demanda este tipo de despliegue haciendo inviable esta solución tanto para el STIC como para el Centro del Cálculo. Se optó por tanto por un despliegue de Hadoop en su versión 2.0 junto con el desarrollo de un sistema de ejecución de tareas automatizado.

4.11 Ejecución automatizada del análisis

Uno de los aspectos críticos del proyecto es la complejidad a la hora de ejecutar tareas de manera correcta debido a la cantidad de pasos que deben ejecutarse en un orden concreto. Lo que en un principio no se incluyó dentro del alcance inicial del proyecto, se convirtió en una necesidad con el fin de dotar de una usabilidad óptima al conjunto del proyecto.

La idea básica desarrollada fue la de aprovechar los tiempos muertos de los recursos hardware con que cuenta el Centro de Cálculo sobre todo en horario no académico para la ejecución automática de la propuesta desarrollada. De esta manera, y de forma transparente para el usuario, un pequeño script de bash se encargaba de limpiar el sistema HDFS, introducir los nuevos inputs a analizar en el sistema HDFS, adjuntar los archivos de configuración adicionales como filtros y datos agregados de soluciones anteriores, y ejecutar el análisis de los datos extrayendo del sistema HDFS el resultado obtenido ya fuese de éxito o de error a un directorio local.

Para automatizar este proceso se hizo uso de cron, un demonio que se encarga de ejecutar procesos en segundo plano a intervalos regulares de Unix. De este modo, todos los días a las 10 de la noche se comprueba el contenido de un directorio local, y en caso de no estar vacío se interpreta como la voluntad de ser analizado. En este punto se comienza la subida de los ficheros a analizar al sistema HDFS y una vez concluido se ejecuta el análisis de los datos. El resultado de este proceso se encuentra en el sistema HDFS, inaccesible para el usuario por lo que una vez generado se mueve al sistema

de ficheros local en el formato requerido y se eliminan todos los ficheros intermedios que se han ido generando así como los ficheros de entrada, dejando la aplicación lista para el siguiente análisis.

4.12 Outputs MapReduce

Convendría analizar la salida del proceso de MapReduce para de esta manera entender y hacerse una idea visual de las modificaciones que se llevan a cabo sobre los datos de entrada, siendo éstos básicamente log de acceso, log DHCP y una serie de filtros o información estática como puede ser ubicación en coordenadas de cada uno de los rangos de ip que se usan internamente.

De esta manera observamos:

- **V0.5 Análisis de una carpeta de ficheros de log**.
  A partir de un conjunto de log de acceso con una estructura como la vista en la Figura 4.2 generamos lo siguiente:

Nombre del fichero	Número de líneas
access.log-20140317	318599
access.log-20140318	527553
access.log-20140319	513896
access.log-20140320	505612
Total de líneas:	1865660

Tabla 4.1. Formato del output, task v0.5.

- **V1.0 Análisis de un fichero de log sencillo un sólo nodo de análisis.**
  Como en el caso anterior, el formato de los ficheros de entrada es el visto en la Figura 4.2 y podemos observar un fragmento del output generado:

Mes	Día	Hora	IP	Número de accesos
Mar	17	10	10.212.2.230	843
Mar	17	10	10.212.2.94	68
Mar	17	10	10.212.3.189	59
Mar	17	10	10.212.3.212	1357
Mar	17	10	10.212.3.233	277

Tabla 4.2. Formato del output, task v1.0

- **V1.5 Análisis de un fichero de log sencillo tres nodos de análisis.**
  El formato de los ficheros de entrada es el visto en la Figura 4.2 y el output generado contiene el mismo formato que el visto en la Tabla 4.2, pero el input en lugar de provenir de un solo log de acceso proviene de 3 ficheros, uno por cada nodo con el fin de comprobar que la agregación entre ficheros funciona correctamente.

- **V2.0 Análisis de varios ficheros de log sencillo de un sólo día en varios nodos de análisis.**
  El output generado en este caso se encarga de realizar un conteo de accesos por IP/fecha/servidor y después ofrecer un resultado total agregado por fecha/IP.

Month	Day	Hour	IP	/udvweb1/Access .log-20140318	/udvweb2/access .log-20140318	/udvweb3/Access .log-20140318	/udvweb4/Access .log-20140318	Total
Mar	17	8	10.219.3.69	0	0	19	0	19
Mar	17	8	10.219.8.238	6	914	0	0	920
Mar	17	8	10.219.8.239	0	1522	0	10	1532
Mar	17	8	10.219.8.242	176	1	0	0	177
Mar	17	8	10.219.8.243	0	0	0	32	32

Tabla 4.3. Formato del output, task v2.0

- **V2.5 Análisis de varios ficheros de log sencillo de un sólo día en varios nodos de análisis con restricciones de día y hora.**
  El formato de los ficheros de entrada es el visto en la Figura 4.2 y el output de este task es idéntico al de la Tabla 4.3 con la salvedad que se introduce un nuevo concepto que es el filtro personalizado en este caso por franja horaria para ignorar el resto de accesos y analizar solo los requeridos en este caso de uso. Dicho filtro cuenta con la siguiente estructura:

  Desde Mar 17 10:00:00
  Hasta Mar 17 17:00:00

- **V3.0 Detectando el origen de las conexiones.**
  El formato de los ficheros de entrada es el visto en la Figura 4.2, teniendo una peculiaridad este task. En este caso se filtra en función de un fichero con rangos de IPs asociados a orígenes lógicos diferentes dentro de la gestión de la red. Este fichero con filtro cuenta con el siguiente contenido:

  192.168.x.x Peligro1
  172.16.x.x Peligro2
  10.x.x.x Interno
  193.145.96.x InternoPublico
  193.145.97.x InternoPublico
  193.145.120.x Administracion
  193.145.125.x InternoPublico
  *.ull.es InternoConDNS
  Resto Externo

  Un fragmento del output generado muestra la siguiente estructura:

Month	Day	Hour	Administracion	Externo	Interno	InternoConDNS	InternoPublico	Peligro1	Peligro2	Total
Mar	17	6	0	4686	103	128	0	0	0	4917
Mar	17	7	0	20003	3495	384	0	0	0	23882
Mar	17	8	0	44578	64334	384	0	0	0	109296
Mar	17	9	0	67733	91041	384	0	0	0	159158

Tabla 4.4. Formato del output, task v3.0

- **V4.0 Enlazando datos no estructurados, correlando elementos en distintos instantes del tiempo.**
  El formato del fichero de entrada es el visto en la Figura 4.3. El objetivo es analizar el fichero de log DHCP con el fin de tener una base sobre la que trabajar para realizar una puesta en común con los datos acceso anteriormente analizados a partir de la tarea V5.5.

Month	Day	Hour	Servidor	Tipo de acceso	IP	Dirección MAC
Mar	18	11:34:32	udvweb1.stic.ull.es	apache2_access	10.225.2.207	00:17:31:c4:30:71

Tabla 4.5. Formato del output, task v4.0

- **V4.5 Desplegando 4.0 para un periodo de log especificado en los filtros de entrada.**
  Esta es una tarea análoga a la V2.5 pero sobre otro conjunto de datos. La estructura de los datos generados responde a la obtenida en la tarea V4.0 pero bajo unos filtros de tiempo.

- **V5.0 Analizando las conexiones WiFi.**
  Similar a la tarea 4.0 pero para conexiones WiFi. El formato de salida es el siguiente:

Date	Hour	MAC	Punto de acceso
01/03/2016	8	88:9f:fa:93:35:7c	10.174.3.28

Tabla 4.6. Formato del output, task v5.0

- **V5.5 Analizando las conexiones WiFi y geoposicionando los accesos.**
  A partir de una serie de ficheros con información relativa al diseño lógico de la red WiFi del STIC generamos una salida con el siguiente

formato para servir como nexo entre los anteriormente desarrollado y el objetivo de la tarea V6.0.

Código del Edificio	Nombre del Edificio	Tipo de red	Máscara de Red	Coordenadas GPS del Edificio
BA2	Bellas Artes Nuevo	Red Azul	10.101.0.0/24	28.461210,-16.276267

Tabla 4.7. Formato del output, task v5.5

- **V6.0 Enlazando todo lo desarrollado.**
  El output de este task fue desde un comienzo el objetivo del proyecto, poder agregar datos de acceso, DHCP y georreferenciarlos con el fin de poder hacer un seguimiento de un dispositivo, análisis de flujos, cargas de la red, etcétera. Las posibilidades son enormes llegados a este punto, siendo la estructura la siguiente:

Date	Punto de Acceso	MAC del dispositivo	Datos agregados sobre ubicación	Coordenadas GPS del Edificio
01/03/2016 8:38	10.174.3.28	88:9f:fa:93:35:7c	PE Periodisco Red WIFI	28.469314,-16.301921
01/03/2016 8:56	10.158.0.207	20:64:32:4c:44:94	IA IUBO+Agricolas Red WIFI	28.481374,-16.319417

Tabla 4.8. Formato del output, task 6.0

4.13 Visualización de los datos obtenidos

El último paso es la correcta visualización de los resultados obtenidos con el fin de convertir simples datos en información útil de cada a la empresa, en este caso el STIC.

Para éste propósito haremos uso del IDE **R-Studio** para R, siendo éste libre y gratuito. **R** es un lenguaje y entorno de programación para análisis estadístico y gráfico muy potente mientras que R-Studio nos facilitará el trabajo con R y la generación de gráficas de manera interactiva.

A su vez sirve para mantener organizado el código dentro de un proyecto y quizás lo más importante, nos permite añadir paquetes de R de manera sencilla y casi automática.

R-Studio en si no lleva a cabo ninguna operación estadística, sino que facilita su realización por parte de R. Permite la lectura directa de los resultados obtenidos a raíz del análisis de datos con Hadoop en formato CSV, creando potentes gráficas 2D para la correcta visualización de los resultados.

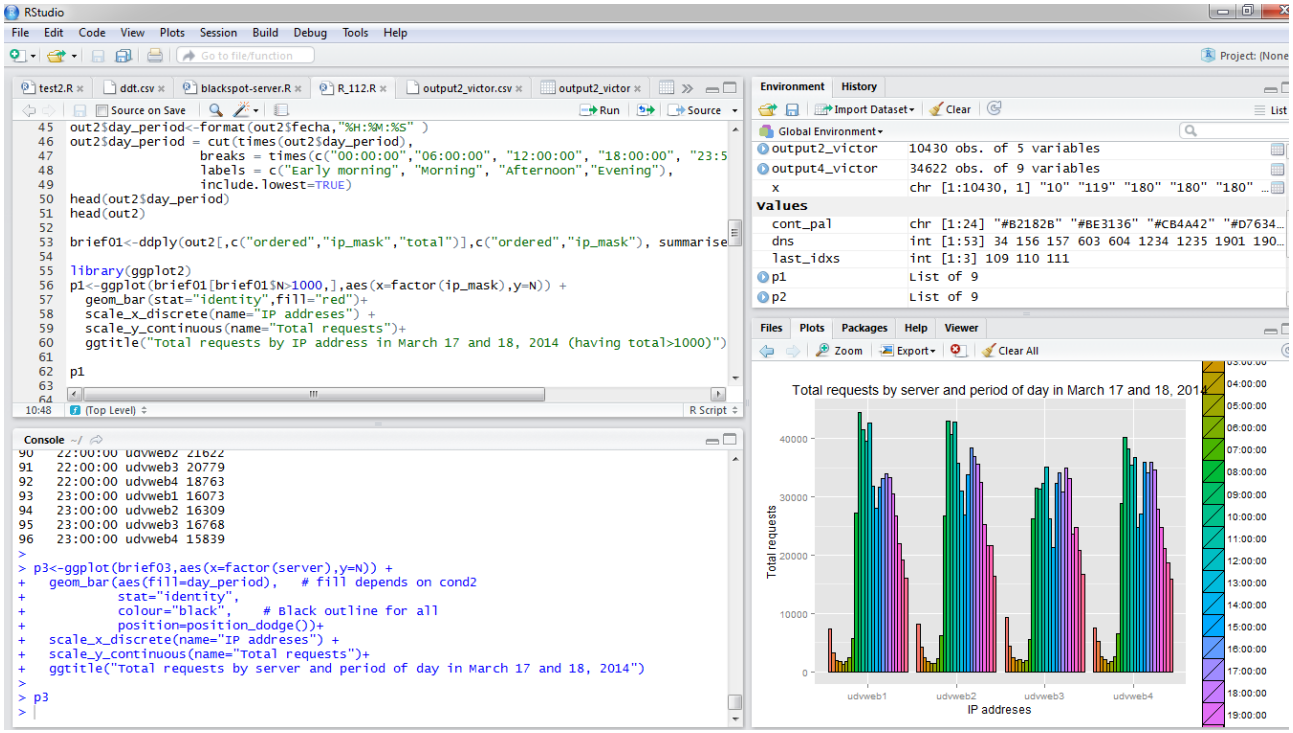

Figura 4.6. Interfaz de R-Studio con pre-visualización de gráfica 2D.

R-Studio permite la publicación directa de resultados en formato HTML para su fácil distribución y cuenta con la potencia suficiente para gestionar grandes cantidades de datos como los que manejamos en este proyecto.

Quizás el aspecto más interesante de R, son las librerías anexas con que cuenta, dándole un potencia visual increíble donde el límite es prácticamente la imaginación de uno mismo. De esta manera, se pueden generar representaciones de datos sobre planos de Google Maps con el fin de georreferenciar los resultados, y añadir ***dashboard*** para dotar de usabilidad y funcionalidad al trabajo realizado.

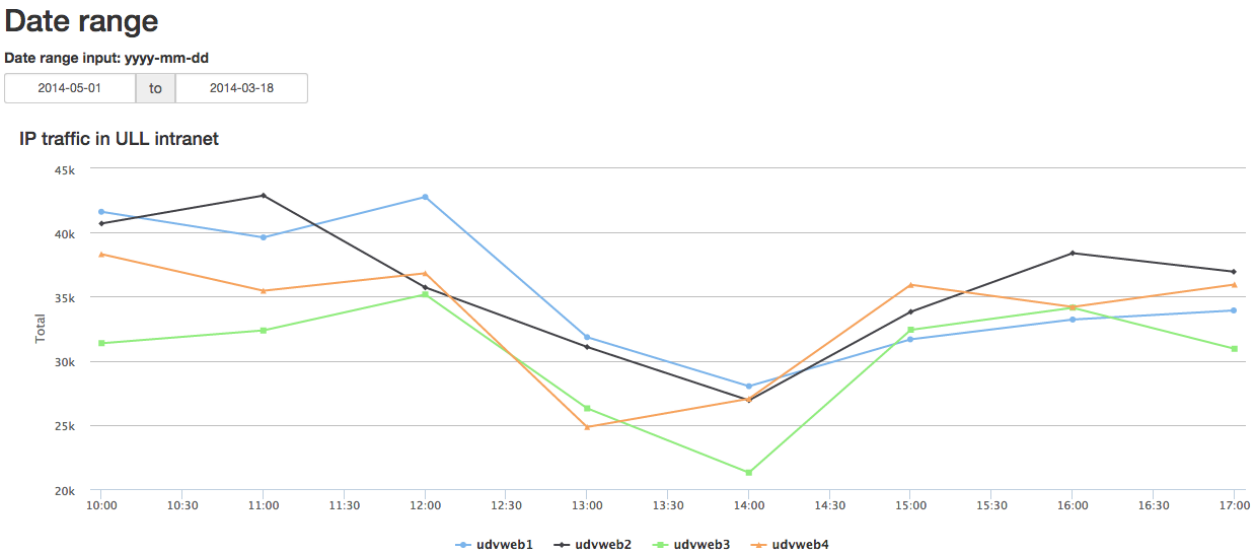

Figura 4.7. Representación 2D con *dashboard* interactivo.

También cabe destacar la potencia de un *framework* que trabaja sobre R-Studio, R-Shiny, especialmente diseñado para la generación de visualizaciones interactivas en la web haciendo uso de librerías de JavaScript como D3, Leaflet o Google Charts, pudiendo obtener resultados tan potentes como el que se muestra en la Figura 4.8.

Sin embargo, esto se sale del ámbito de nuestro proyecto y quizás sería una excelente línea para el desarrollo de futuros Trabajos de Fin de Grado basados en la visualización y representación de datos masivos.

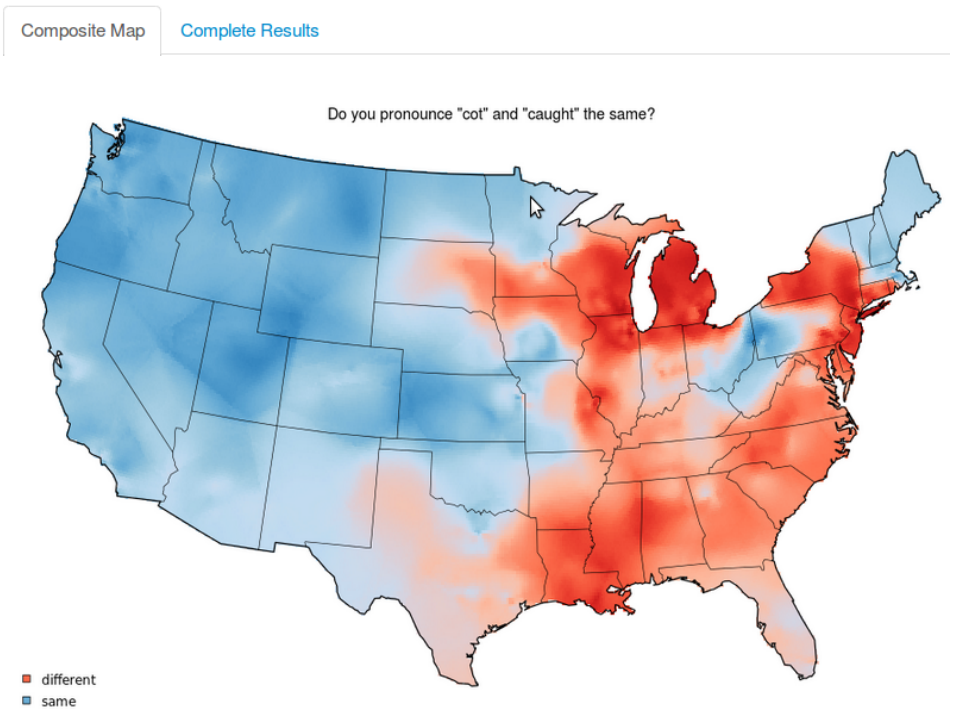

Figura 4.8. Ejemplo de representación georreferenciada.

Capítulo 5.
Conclusiones y líneas futuras

Big Data es el futuro. Puede que tal afirmación suene pretenciosa sobre todo teniendo en cuenta la infinidad de tecnologías que se desarrollan en la actualidad, pero cada vez más el reto que supone el análisis de unos datos que aumentan en volumen de manera exponencial y los beneficios derivados del correcto tratamiento de los mismos, hace que Big Data sea a día de hoy una de las tecnologías con mayor proyección a nivel mundial.

La situación en el marco español es ligeramente diferente, porque si bien en Estados Unidos, Alemania, o Francia la tecnología del análisis de grandes cantidades de datos se encuentra en pleno auge o "*hype*", en España se están dando los primeros pasos en ese sentido de la mano de grandes empresas como BBVA y Banco Santander. Esto supone un desfase de alrededor de dos años con respecto a los países punteros en este ámbito por lo que representa una oportunidad de presente en el extranjero o a corto plazo a nivel nacional.

Al ser un proyecto multidisciplinar las conclusiones obtenidas son diversas, más incluso cuando no se ha limitado el alcance del proyecto a un desarrollo meramente académico, sino que se le ha otorgado de un valor extra al realizarse de manera colaborativa con una entidad externa tal como el STIC de La Laguna.

Siguiendo esta última idea, la primera conclusión que se obtiene es el enriquecimiento del proyecto que se ha conseguido al darle un enfoque eminentemente práctico, con datos reales y elaborando una solución compacta a este problema.

El hecho que de los datos de entrada sean reales añade un componente extra como es la variabilidad de los mismos, pues los log analizados recogen de igual manera errores, presentan pequeñas discrepancias en factores como la hora de cada entrada derivado del tiempo que invierte el software de generación de log en su recogida y procesado, etc.

Esto no hace más que enriquecer la solución puesto que obliga a generar el código mucho más compacto que integre una cierta tolerancia a errores.

De forma paralela sirve como puente entre el ámbito académico y mundo empresarial convirtiéndose en una primera piedra de toque a la hora de realizar proyecto en cierta medida colaborativo con terceros, dependiendo en muchos momentos de ellos para el correcto devenir del mismo. A pesar de que en ocasiones ha podido resultar frustrante el tiempo de espera entre reuniones debido a la carga de trabajo propia de una entidad como el STIC que ha retrasado el avance del proyecto, el contar con gente tan preparada y con tanta experiencia lo compensa con creces.

El desarrollo del proyecto no ha hecho más que afianzar mi idea inicial de que el mundo de Big Data presenta un futuro muy prometedor y que se encuentra en un momento excepcional para adentrarse en el mismo, con una tecnología sólida y contrastada como es Hadoop MapReduce y la aparición de nuevos *frameworks* de desarrollo como puede ser principalmente Spark, si bien esto no quita la dificultad que este proceso entraña.

La curva de aprendizaje de una tecnología como Hadoop no es sencilla, y una tarea tan básica como la instalación básica del *framework* en un sistema mononode por primera vez incluye no pocos retos y desafíos. Si entramos a valorar el despliegue de la solución en un sistema multinode para el análisis de datos de manera distribuida la dificultad aumenta exponencialmente.

Por otra parte, y hablando del propio desarrollo llevado a cabo, cabe destacar las increíbles capacidades y posibilidades que ofrece el ecosistema Hadoop para los amantes del Big Data, tanto por las opciones en cuanto al lenguaje utilizado para su desarrollo como puede ser Java, Python o C++ o la cantidad de elementos tangenciales tales como librerías, servicios y APIs de gran calidad y potencia.

Una de estas librerías, Streaming, nos fue en cierta medida impuesta de manera que el desarrollo fuese llevado a cabo mediante lenguaje Python, y si bien al comienzo supuso un problema debido a la falta de conocimiento sobre el mismo y a la poca documentación de la librería en comparación a la solución nativa, el objetivo pudo cumplirse.

En cuanto al apartado visual se optó por un acercamiento al tema a través del lenguaje de análisis estadístico R. La potencia del mismo es increíble, el límite está prácticamente en el resultado que uno espere conseguir.

De hecho, ésta podría ser una futura línea de investigación, trabajar en la visualización de datos masivos haciendo uso de R. En este caso debido a que los datos se encuentran georreferenciados se podrían plasmar sobre un mapa y ofrecer una visualización interactiva de los mismos. Esto requeriría de un nivel medio de conocimiento en JavaScript y HTML.

A su vez, si surgen nuevos requerimientos de análisis de datos por parte del STIC, o si la morfología de los log de acceso cambia, se podría llevar a cabo una actualización de la solución desarrollada.

Capítulo 6.
Summary and Conclusions

Big Data is the future. This might sound like a pretentious statement, especially bearing in mind the countless number of technologies currently being developed. However, the challenge posed by the analysis of data, increasing in volume at an exponential rate, and the benefits stemming from the correct processing of that data, makes Big Data currently one of the technologies with the greatest impact on a global level.

The situation in Spain is slightly different, because while in the USA, Germany or France, the technology of analysing large quantities of data is currently in a boom, in Spain, the technology is making its first steps in that direction being led by large companies, such as BBVA and Santander Bank. This suggests a lag of about two years with respect to the leading countries in this field, and indicates an opportunity abroad or in the short term on a national level.

Being a multi-disciplinary project, the conclusions obtained are diverse, even more when the project reach has not been limited to a merely academic undertaking, but rather it has gained added value by being undertaken collaboratively with an external partner such as the ICT services at the University of La Laguna.

Continuing with this last idea, the first conclusion that was obtained is the enrichment of the project, thanks to the completely practical character, with real data and formulating a compact solution to this problem.

The fact that the input data were real adds an extra component such as the variability of data, as the analysed logs also contained errors, showing slight discrepancies in factors such as the time of each entry resulting from the time that the log generation software puts in its collection and processing, etc.

This only enriches the solution, inasmuch that it forces the generation of a code which is much more compact and that allows a certain tolerance of error.

Alongside that, it serves as a bridge between the academic environment and the world of business, converting itself into the first touchstone at the time of carrying out the project with a certain measure of collaboration with third parties, depending on them at many times for the correct outcome of the project. Despite the delay in the advancement of the project because of the waiting time between meetings (due to the normal workload for an organisation such as the ICT services), relying on people who are so prepared and who have so much experience resulted in being extremely rewarding.

The development of the project has only strengthened my initial idea that the world of Big Data presents a promising future and it is in an exceptional moment to embark on it, with a solid and contrasting technology such as Hadoop MapReduce; as well as the appearance of new frameworks of development such as Spark, although this does not subtract from the difficulty that this process involves.

The learning curve for a technology such as Hadoop is not simple, and such a simple task as the basic installation of the framework in a standalone system for the first time includes many challenges. If we start to evaluate the deployment of the solution in a multinode system for the analysis of data in a distributed way, the difficulty increases exponentially.

On the other hand and talking about the exploration carried out, it is worth noting the incredible capacities and possibilities that the Hadoop ecosystem offers for Big Data lovers; for both the options in the language used for its development, such as Java, Python or C++ and the quantity of tangential elements such as libraries, services and APIs of high quality and power.

One of those libraries, Streaming, was imposed on us, so the development was carried out using the Python language, and even though at the start it was a problem due to the lack of knowledge and the little documentation about the library en comparison with the native solution, the aim could be achieved.

In reference to the visualisation, it was decided to opt for an approach through the statistical analysis language R. The power of this tool is incredible and its possibilities are boundless.

In fact, this could be a future line of research, making use of R in the working with the mass data visualisation. In this case, due to the fact that the data were georeferenced, it was possible to plot them on a map and offer an interactive visualisation of the data. This would require an intermediate knowledge of JavaScript and HTML.

At the same time, if new requirements for the analysis of data arise from the ICT services, or if the morphology of the access logs changes, an update of the solution that was developed could be carried out.

Capítulo 7. Presupuesto

En este proyecto, la cantidad de hardware necesario depende de las necesidades y aspiraciones de rendimiento del STIC, pudiendo desplegarse en un solo dispositivo virtualizando nodos, o dedicar al número de nodos que se desee debido a la fácil escalabilidad que ofrece Hadoop. Desde un comienzo, el objetivo era utilizar los tiempos muertos de cómputo de los dispositivos con los que ya contaba el STIC sobre todo fuera de horario laboral por lo que la inversión en este aspecto debería ser nula.

7.1 Recursos Hardware.

En caso de carecer de una infraestructura previa sobre la que realizar el despliegue del clúster distribuido, la inversión requerida puede variar mucho en función de las tareas que se vayan a llevar a cabo y del rendimiento que se espere conseguir. En función del nivel objetivo (no es lo mismo una instalación con afán académico que otra con afán empresarial) podemos encontrarnos con una inversión de 7.000 € en un HP ProLiant SL4500 Gen8 (diseñado para procesos de Big Data) hasta una solución mucho más económica en torno a los 2.000 €.

7.2 Recursos Software y Licencias

Otra de las premisas del proyecto era la utilización únicamente de software libre por lo que el gasto en licencias es cero, pudiendo verse modificado en función del IDE de desarrollo que se desee usar.

7.3 Recursos Humanos

La instalación y configuración junto con el desarrollo de los scripts necesarios para el procesamiento de datos supone una fuerte inversión de tiempo, por lo que si consideramos la contratación de personal cualificado por un salario de 35€ / hora, y una duración estimada de 150 horas de trabajo mínimo obtenemos unos requisitos de 5.250 €.

Apéndice A. Ejemplos de código desarrollado

A continuación añadiré unos pocos scripts desarrollados durante el desarrollo del proyecto, tanto mapper() como reducer() para ilustrar la estructura de los mismos.

A.1. Script Mapper1.py

```
/*******************************************************************************
*
* Mapper1.py
*
*******************************************************************************
*
* AUTOR: Víctor Plaza Martín
*
* FECHA: Marzo de 2015
*
* DESCRIPCION:
*   #!/usr/bin/env python
*   __author__ = 'victor'
*
*   import sys
*   import re
*   import os
*
*   def read_input(file):
*          for line in file:
*                 try:
*                        input_file = os.environ['map_input_file']
*                 except KeyError:
```

```
*                       input_file = os.environ['mapreduce_map_input_file']
*               a, b = os.path.split(input_file)
*               custom_input = "/" + os.path.basename(a) + "/" + b + " " + str(line)
*               yield custom_input
*
*
*   def main():
*         parts = [
*               r'(?P<path>\S+)',
*               r'(?P<month>\S+)',
*               r'(?P<day>\S+)',
*               r'(?P<hour>\S\S)\S+',
*               r'\S+',
*               r'\S+',
*               r'(?P<ip>\S+)',
*               r'.+',
*         ]
*         pattern = re.compile(r'\s+'.join(parts)+r'\s*\Z')
*
*         data = read_input(sys.stdin)
*         for line in data:
*               res = pattern.match(str(line)).groupdict()
*               res2 = res["month"] + '\t' + res["day"] + '\t' + res["hour"] + '\t'
*                     + res["ip"] + '\t\t' + res["path"]
*               sys.stdout.write('%s\t\t%d\n' % (res2, 1))
*
*   if __name__ == "__main__":
*         main() *
**********************************************************************************/
```

A.2. Script Mapper2.py

```
/*******************************************************************************
*
* Mapper2.py
*
*******************************************************************************
*
* AUTORES: Víctor Plaza Martín
*
*
* FECHA: Abril de 2015
*
*
* DESCRIPCION
*   #!/usr/bin/env python
*   __author__ = 'victor'
*
*   import sys
*   import re
*   import os
*
*
*   def read_input(file):
*       for line in file:
*           try:
*               input_file = os.environ['mapreduce_map_input_file']
*           except KeyError:
*               input_file = os.environ['map_input_file']
*           a, b = os.path.split(input_file)
*           custom_input = "/" + os.path.basename(a) + "/" + b + " " + str(line)
*           yield custom_input
*
*   def between(month, day, hour):
*       if beg[1] == month == end[1] and beg[2] <= day <= end[2] and beg[3] <= hour
*          <= end[3]:
*           return True
*       return False
*
*   def setlimit():
```

```
*       global beg
*       beg = []
*       global end
*       end = []
*       with open("filtro.txt", 'r') as f:
*           for line in f:
*               if not beg:
*                   beg = line.split()
*               else:
*                   end = line.split()
*
*
*   def main():
*       parts = [
*           r'(?P<path>\S+)',
*           r'(?P<month>\S+)',
*           r'(?P<day>\S+)',
*           r'(?P<hour>\S+)',
*           r'\S+',
*           r'\S+',
*           r'(?P<ip>\S+)',
*           r'.+',
*       ]
*       pattern = re.compile(r'\s+'.join(parts)+r'\s*\Z')
*       setlimit()
*       data = read_input(sys.stdin)
*       for line in data:
*           res = pattern.match(str(line)).groupdict()
*           if between(res["month"], res["day"], res["hour"]):
*               res2 = res["month"] + '\t' + res["day"] + '\t' + res["hour"][:2]
*                       + '\t' + res["ip"] + '\t\t' + res["path"]
*               sys.stdout.write('%s\t\t%d\n' % (res2, 1))
*           else:
*               continue
*
*   if __name__ == "__main__":
*       main()
**************************************************************************/
```

A.3. Script Mapper3.py

```
/*******************************************************************************
*
* Mapper3.py
*
*******************************************************************************
*
* AUTORES: Víctor Plaza Martín
*
*
* FECHA: Junio de 2015
*
*
* DESCRIPCION
*   #!/usr/bin/env python
*   __author__ = 'victor'
*
*   import sys
*   import zipimport
*   importer = zipimport.zipimporter('net.mod')
*   netaddr = importer.load_module('netaddr')
*   from netaddr.ip import IPAddress, IPNetwork, all_matching_cidrs
*
*   def read_input(file):
*       for line in file:
*           yield line
*
*   def main():
*       list = []
*       global result
*       result = {}
*       with open("IPS.txt", 'r') as f:
*           for line in f:
*               if line != "":
*                   aux = line.split(';')
*                   result.setdefault(str(aux[1]), {})[str(aux[0])
*                             .replace('\t', ' ')] = str(aux[2]).replace('\n', '')
*                   if str(aux[1]) not in list:
*                       list.append(str(aux[1]))
```

```
*       data = read_input(sys.stdin)
*       for line in data:
*           res = line.split(';')
*           try:
*               res2 = all_matching_cidrs(res[1], list)
*               if res2 != '[]':
*                   sys.stdout.write('%s;%s\n' % (line.replace('\n',''),
*                      str(result[str(res2).split("'")[1]]).split("'")[1]+ ';' +
*                      str(result[str(res2).split("'")[1]]).split("'")[3]))
*           except:
*               sys.stdout.write('%s;%s\n' % (line.replace('\n',''),
*                            "SIN COINCIDENCIA"))
*
*
*   if __name__ == "__main__":
*         main()
*********************************************************************************/
```

A.4. Script Reducer1.py

```
/*******************************************************************************
*
* Reducer1.py
*
*******************************************************************************
*
* AUTORES: Víctor Plaza Martín
*
*
* FECHA: Marzo 2015
*
*
* DESCRIPCION
*   #!/usr/bin/env python
*   __author__ = 'victor'
*
*   import sys
*   import os
*
*   current_path = None
*   path = None
*   filepath = None
*   filename = None
*   current_count = 0
*   total_count = 0
*   dic = {}
*
*   for line in sys.stdin:
*
*       line = line.strip()
*       path, count = line.split('\t\t', 1)
*
*       if current_path == path:
*           current_count += int(count)
*       else:
*           if current_path:
*               filepath, filename = os.path.split(current_path)
*               dic[filename] = current_count
```

```
*               #dic[str(current_path)] = current_count
*               total_count += current_count
*           current_count = int(count)
*           current_path = path
*
*   if current_path == path:
*       filepath, filename = os.path.split(current_path)
*       dic[filename] = current_count
*       #dic[str(path)] = current_count
*       total_count += current_count
*
*
*   for k, v in dic.items():
*       sys.stdout.write(k + '\t' + str(v) + '\n')
*   sys.stdout.write("Total de lineas:" + '\t' + str(total_count) + '\n')
*
*****************************************************************************/
```

A.5. Script Reducer2.Py

```
/*******************************************************************************
*
* Mapper2.py
*
*******************************************************************************
*
* AUTORES: Víctor Plaza Martín
*
*
* FECHA: Abril 2015
*
*
* DESCRIPCION
*   #!/usr/bin/env python
*   __author__ = 'victor'
*
*   import sys
*
*   def read_mapper_output(file, separator='\t\t'):
*       for line in file:
*           yield line.rstrip().split(separator, 2)
*
*   def main(separator='\t\t'):
*       data = read_mapper_output(sys.stdin, separator=separator)
*       diffiles = set()
*       difip = set()
*       result = {}
*       output = ""
*       cabecera = ""
*       sum = 0
*
*       for ip, path, count in data:
*           try:
*               result.setdefault(ip, {})[path] += int(count)
*               difip.add(ip)
*               diffiles.add(path)
*           except:
*               result.setdefault(ip, {})[path] = 1
```

```
*               difip.add(ip)
*               diffiles.add(path)
*
*       for x in sorted(diffiles):
*           cabecera += x + '\t'
*
*       print("Month\tDay\tHour\tIp\t", cabecera, "Total")
*       for x in sorted(difip):
*           for y in sorted(diffiles):
*               try:
*                   output += str(result[x][y]) + '\t'
*                   sum += result[x][y]
*               except:
*                   output += str(0) + '\t'
*           print(x, '\t', output, sum)
*           output = ""
*           sum = 0
*
*   if __name__ == "__main__":
*       main()
**********************************************************************************/
```

A.6. Script Reducer3.py

```
/*******************************************************************************
*
* Reducer3.py
*
*******************************************************************************
*
* AUTORES: Víctor Plaza Martín
*
*
* FECHA: Junio 2015
*
*
* DESCRIPCION
*   #!/usr/bin/env python
*   __author__ = 'victor'
*
*   import sys
*
*
*
*   def read_mapper_output(file, separator='\t\t'):
*       for line in file:
*           yield line.split(separator, 1)
*
*   def main(separator='\t\t'):
*       auxip = ""
*       auxmac = ""
*       data = read_mapper_output(sys.stdin, separator=separator)
*
*       for datos, valor in data:
*           ip, hora, mac = datos.split(';')
*           if str(ip) != auxip or str(mac) != auxmac:
*               print(ip + '\t' + hora + '\t' + mac)
*               auxip, auxmac = ip, mac
*
*   if __name__ == "__main__":
*       main()
*******************************************************************************/
```

Bibliografía

- Wiki de la librería Streaming.
  hadoop.apache.org/docs/current/hadoop-streaming/HadoopStreaming.html
- Wikipedia. es.wikipedia.org
- Wiki Hadoop. wiki.apache.org/hadoop/
- Vignesh Prajapati. Noviembre de 2013. Big Data Analytics with R and Hadoop.
- Tom White, Editorial O'Really (2012). Hadoop – The Definitive Guide.
- Big Data. www.ibm.com/developerworks/ssa/local/im/que-es-big-data/
- Hadoop MapReduce vs Apache Spark.
  www.xplenty.com/blog/2014/11/apache-spark-vs-hadoop-mapreduce
- Hadoop desarrollado mediante Python.
  blog.cloudera.com/blog/2013/01/a-guide-to-python-frameworks-for-hadoop
- Instalar un clúster mononode de Hadoop.
  www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-single-node-cluster
- Filosofía Hadoop MapReduce.
  datosintensos.blogspot.com.es/2012/07/hadoop-y-la-filosofia-en-un-sistema.html
- Desarrollar nuestra primera aplicación con Python.
  www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python
- Mejorar el rendimiento de la solución desarrollada.
  wiki.python.org/moin/PythonSpeed
- Órdenes de complejidad en los operadores Python.
  www.ics.uci.edu/~pattis/ICS-33/lectures/complexitypython.txt
- Tutorial básico de ejecución MapReduce en Python.
  blog.matthewrathbone.com/2013/11/17/python-map-reduce-on-hadoop---a-beginners-tutorial.html
- R-Shiny. shiny.rstudio.com
- 3 V's Big Data.

http://www.lantares.com/blog/velocidad-variedad-y-volumen-las-3-magnitudes-clave-de-big-data

Complejidad de los datos. https://datafloq.com/read/3vs-sufficient-describe-big-data/166

Yarn – Apache Hadoop.

http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html