ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

"Sistema para Generar Gráficas a Partir de Logs Tcpdump
Usando Hadoop"

INFORME DE MATERIA DE GRADUACIÓN

Previa a la obtención del Título de:

INGENIERO EN COMPUTACIÓN ESPECIALIZACIÓN SISTEMAS DE INFORMACIÓN

Presentada por:

ANGEL STALIN CRUZ PALAQUIBAY
PEDRO ALFREDO TORRES ARELLANO

GUAYAQUIL – ECUADOR

2009

AGRADECIMIENTO

Agradecemos a Dios, a nuestras familias, profesores y amigos que nos proporcionaron fuerzas, ayuda, conocimiento y apoyo para seguir adelante. A la MSc. Cristina Abad por su apoyo y colaboración para la culminación de este proyecto.

DEDICATORIA

A Dios

A mis padres

A mis familiares

A mis amigos

TRIBUNAL DE SUSTENTACIÓN

MSc. Cristina Abad
PROFESOR DE LA MATERIA

Ing. Juan Moreno
PROFESOR DELEGADO DEL DECANO

DECLARACIÓN EXPRESA

"La responsabilidad del contenido de esta Tesis de Grado, nos corresponden exclusivamente; y el patrimonio intelectual de la misma a la ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL"

(Reglamento de Graduación de la ESPOL)

____
Ángel Cruz Palaquibay

____
Pedro Torres Arellano

RESUMEN

El propósito de este proyecto de materia de graduación es la de realizar un sistema que genere gráficas a partir de un procesamiento y análisis de logs tcpdump. El tamaño de los Logs a procesar es relativamente grande, por lo que se necesita un procesamiento masivo de estos datos, para ello utilizamos el esquema MapReduce a través del Framework Hadoop.
En el capítulo 1, se analiza los datos a ser procesados por el sistema, se realiza una breve descripción del formato de Logs así como la conversión de los mismos.

En el capítulo 2, se plantea el diseño de la solución, las herramientas que utilizamos para desarrollar el sistema, además explica cómo trabaja las fases MapReduce en nuestra solución.

En el capítulo 3, se detalla la implementación y el funcionamiento del sistema, se describe las pruebas realizadas en los servicios web de Amazon así como los resultados obtenidos.

Finalmente en el capítulo 4, se presentan las conclusiones obtenidas con el desarrollo de sistema con el esquema MapReduce. En las recomendaciones se sugiere herramientas complementarias que podrían ayudar a un mejor análisis de Logs.

ÍNDICE GENERAL

Pág.

ÍNDICE DE FIGURAS

Pág.

ÍNDICE DE TABLAS

Pág.

INTRODUCCIÓN

Los ataques constantes de spammers o crackers, hacia redes reales, han puesto en marcha la conformación de mecanismos que ayuden de alguna manera a reducir estos posibles ataques.

Uno de estos mecanismos son las denominadas honeypots de alta interacción o Honeynet, los cuales resultan ser anzuelos para posibles ataques que provienen del exterior [1]. El estudio y análisis de estos ataques se convierten en una ventaja de seguridad, al tener conocimiento de la forma de actuar dichos intrusos, se podrán desarrollar políticas que apunten a tener un menor impacto en las redes reales.

Estos ataques son monitoreados en la red, que a su vez se van almacenando en registros o logs, formando una base de datos del tráfico. Estos datos masivos guardados en los logs, pueden ser de mucha ayuda, si se obtienen conocimiento de ellas a través de diferentes tipos de gráficos.

La complejidad del tratamiento de estos datos masivos, hacen que su procesamiento se vea afectado. Una alternativa es el paradigma MapReduce que permite realizar de manera eficiente este procesamiento masivo de logs, logrando de esta manera abarcar grandes cantidades de datos y poderlos representar gráficamente.

CAPÍTULO 1

1. ANÁLISIS DE DATOS

El presente capítulo describe el dataset que se utilizó para la generación de reportes gráficos. Se exponen las características del dataset, el proceso de análisis de su contenido, así como también las herramientas usadas para la conversión de los mismos.

1.1. Definición dataset

Un dataset [2] es un conjunto de información asociada a una fuente de datos. Esta información son detalles de sus características generales (extensión, tablas asociadas, características de sus datos, etc.).

1.2. Descripción de logs pcap

Los logs pcap son archivos que contienen una serie de registros del tráfico de red capturado, estos logs son los que utilizamos como datos de entrada para nuestro proyecto. Para el caso de las capturas de una Honeynet estos logs suelen ser de gran tamaño en el orden de Gigabytes o más.

Los logs pcap guardan información tales como las fecha en la que se envió un paquete, direcciones MAC, direcciones IP, puertos, protocolos, data del paquete, etc.

```
13:08:05.737768 ppp0 > slip139-92-26-177.ist.tr.ibm.net.1221 > dsl-usw-cust-110.inetarena.com.www: . 342:342(0) ack 1449 win 31856 <nc
,nop,timestamp 1247771 114849487> (DF)
13:08:07.467571 ppp0 < dsl-usw-cust-110.inetarena.com.www > slip139-92-26-177.ist.tr.ibm.net.1221: . 1449:2897(1448) ack 342 win 3185€
<nop,nop,timestamp 114849637 1247771> (DF)
13:08:07.707634 ppp0 < dsl-usw-cust-110.inetarena.com.www > slip139-92-26-177.ist.tr.ibm.net.1221: . 2897:4345(1448) ack 342 win 3185€
<nop,nop,timestamp 114849637 1247771> (DF)
13:08:07.707922 ppp0 > slip139-92-26-177.ist.tr.ibm.net.1221 > dsl-usw-cust-110.inetarena.com.www: . 342:342(0) ack 4345 win 31856 <nc
,nop,timestamp 1247968 114849637> (DF)
13:08:08.057841 ppp0 > slip139-92-26-177.ist.tr.ibm.net.1045 > ns.de.ibm.net.domain: 8928+ PTR? 110.107.102.209.in-addr.arpa. (46)
13:08:08.747598 ppp0 < dsl-usw-cust-110.inetarena.com.www > slip139-92-26-177.ist.tr.ibm.net.1221: P 4345:5793(1448) ack 342 win 3185€
<nop,nop,timestamp 114849813 1247968> (DF)
```

Figura 0.1 Ejemplo de un log pcap convertido a texto plano

1.3. Conversión de logs pcap

En la actualidad existen programas que manejan logs pcap. Entre ellos encontramos a Wireshark [3]. Estos programas hacen uso de librerías diseñadas para cada entorno de programación, es así como hay librerías para Java, C++, Visual Basic, entre otros. Lastimosamente, esos programas cargan los datos a ser analizados en memoria, y por lo tanto no pueden analizar archivos del orden de varios Gigabytes o superior.

En el presente proyecto se transformó los logs pcap a texto plano para poderlos usar de manera natural en la plataforma de procesamiento distribuido utilizada (Hadoop).

Nuestro trabajo está realizado en el lenguaje de programación Java, utilizando la librería JNetStream v2.4.

1.3.1. JNetStream

JNetStream [4] es una librería de Java que nos permite capturar y enviar paquetes. Además se pueden desarrollar aplicaciones que capturen paquetes desde una interface de red, visualizarlos y analizarlos desde Java. La mayoría de los funciones de JNetStream están en el área de decodificar la estructura de los paquetes que están en representación binaria.

La librería JNetStream ha sido probada para Microsoft Windows (98/2000/XP/Vista), Linux (Fedora, Mandriva, Ubuntu), Mac OS X, FreeBSD, and Solaris, con lo cual se cubre un amplio espectro de sistemas operativos a utilizar.

1.3.2. Formato de conversión de logs

Para el presente proyecto se utilizó la librería JNetStream para convertir los archivos pcap que están en binario al siguiente formato de texto plano:

Año, mes, día, hora (hh:mm:ss), Ip_fuente, ip_destino, puerto_fuente, puerto_destino, protocolo

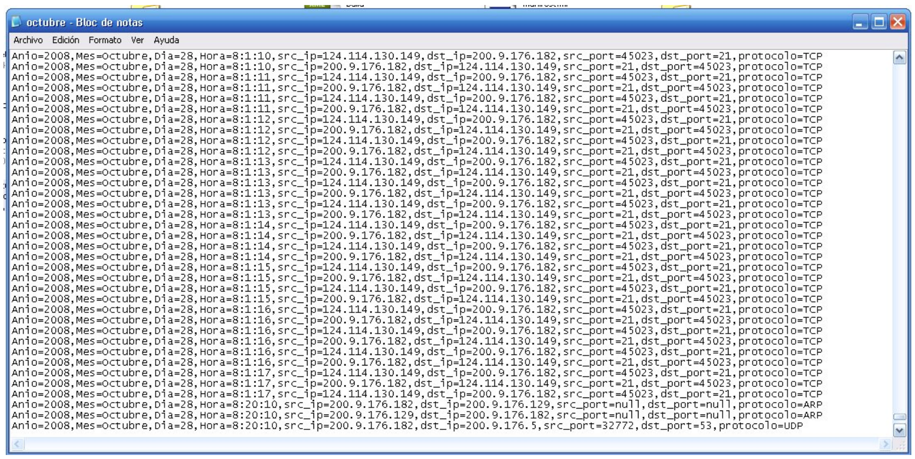


Figura 0.2 Ejemplo del formato de conversión de logs pcap a texto

En la Figura 1.2 observamos el formato final a la que los archivos pcap fueron transformados. Esta conversión se realizó debido a que su formato binario dificultaba de gran manera la manipulación en Hadoop. Es por eso que optamos por realizar una previa conversión de los archivos pcap a archivos de texto plano, el cual nos beneficia en el sentido de que ahora podemos manipular los datos de los paquetes dándoles un formato específico para nuestros intereses. No obstante realizar este proceso de conversión conllevaría más tiempo que tratar de procesar directamente con los archivos pcap.

CAPÍTULO 2

2. DISEÑO DE LA SOLUCIÓN AL PROBLEMA

En el presente capítulo se explica todo lo referente al diseño de nuestra solución para realizar las gráficas a partir de Logs pcap. Se describe el diseño general utilizando los servicios de Amazon Web Services y el diseño específico utilizando el paradigma MapReduce usando Hadoop.

2.1. Introducción paradigma MapReduce

MapReduce es un paradigma de programación propuesto por Google [5] cuya implementación ha sido realizada para dar soporte a la computación paralela sobre grandes colecciones de datos en grupos de computadores (clústeres). Posterior a la publicación del trabajo de Google, han surgido varias otras implementaciones de MapReduce, utilizando diversos lenguajes de programación. En el presente proyecto utilizamos la versión libre con mayor aceptación en la actualidad: Apache Hadoop [6].

Para la implementación de MapReduce es necesario especificar dos funciones como son *Map* y *Reduce*.

2.1.1. Map

La función Map o Mapeo recibe un ítem de datos de entrada en forma de pares (k1, v1) y produce una lista de valores pares intermedios (k2, v2) por cada llamada a esta función. Posteriormente junta todos los valores que tengan una misma clave y los agrupa teniendo una lista de valores por cada clave.

Map (k1, v1) -> list (K2, v2)

2.1.2. Reduce

La función Reduce recibe la lista de valores y claves enviadas por el Mapeo, produce una colección de valores para cada dominio. Esto lo hace en cada llamada al Reduce, el retorno de todas esas llamadas se recoge como la lista de resultado deseado.

Reduce (K2, list (v2)) -> list (v2)

2.2. Framework Hadoop

Hadoop es una plataforma que nos permite desarrollar aplicaciones distribuidas que tengan que tratar con grandes cantidades de datos del orden de los Peta bytes. Hadoop provee escalabilidad además de trabajar muy bien en un clúster de servidores, ofrece ventajas de la programación distribuida aunque eso no signifique que el programador tenga que preocuparse por el paralelismo y tolerancia a fallos de las aplicaciones que desarrolla ya que Hadoop lo implementa.

2.3. Framework para las graficas – JFreeChart

JFreeChart [7] es una librería para gráficos escrita en Java que facilita generación de gráficos de calidad profesional en nuestras aplicaciones, ya sean Web o de escritorio. Entre sus características principales tenemos:

- Un API consistente y bien documentado con soporte para un amplio rango de tipos de gráficas.
- Un diseño flexible fácilmente extendible, y la posibilidad de ser usado tanto en tecnologías de servidor (aplicaciones Web) y de Cliente (Swing, por ejemplo).

- JFreeChart está distribuido bajo la licencia LGPL.

2.4. Diseño, estructura de los reportes (gráficos a generar)

Para la generación de reportes gráficos nos basamos en los puertos de mayor importancia, así como los protocolos más comunes como son tcp, udp entre otros [8]. A continuación se describen los tipos de reporte que serán generados por el sistema.

2.4.1. Gráfica de las 10 ips destinos más concurridos

Para este gráfico se requiere el siguiente formato:

IP Destino	***# de concurrencia***
200.198.34.54	37644
192.170.38.45	1290
198.114.123.90	1218

2.4.2. Gráfico de las 10 ips fuentes que más visitaron

Para este gráfico se requiere el siguiente formato:

IP Fuente	***# de concurrencia***
200.198.34.54	37644
192.170.38.45	1290
198.114.123.90	1218

2.4.3. Actividad por Protocolos TCP- UDP – ICMP

Para este gráfico se requiere el siguiente formato:

Tipo protocolo	***# de concurrencia***

TCP	157381
UDP	47705
ICMP	3134

2.4.4. Actividad por Puertos TCP y UDP

Para este gráfico se requiere el siguiente formato:

Protocolo TCP	***# de concurrencia***
microsoft-ds (445)	7661
ircd (6667)	4742
ssh (22)	568
http (80)	334
ftp (21)	34

Protocolo UDP	***# de concurrencia***
netbios-ns (137)	1770
138 (netbios-dgm)	138
1434 (ms-sql-m)	155

2.4.5. Gráfico de los paquetes por día del mes de Agosto

Para este gráfico se requiere el siguiente formato:

Mes /	***día***	***# de concurrencia***
Agosto	1	1435
Agosto	2	1290
.	.	.
.	.	.
Agosto	31	293

2.4.6. Gráfico de los paquetes por día del mes de Septiembre

Para este gráfico se requiere el siguiente formato:

Mes	***día***	***# de concurrencia***
Septiembre	1	1435
Septiembre **.** **.** **Septiembre**	2 . . 30	1290 . . 293

2.4.7. Gráfico de los paquetes por día del mes de Octubre

Para este gráfico se requiere el siguiente formato:

Mes	***día***	***# de concurrencia***
Octubre	1	1435
Octubre **.** **.** **Octubre**	2 . . 15	1290 . . 293

2.4.8. Gráfico de los paquetes por día del mes de Noviembre

Para este gráfico se requiere el siguiente formato:

Mes	***día***	***# de concurrencia***
Noviembre	1	1435
Noviembre **.** **.** **Noviembre**	2 . . 15	1290 . . 293

2.5. Diseño general con AWS

Para el desarrollo de nuestra aplicación utilizamos los servicios de Amazon Web Services que permiten, entre otras cosas, levantar clústeres Hadoop bajo demanda. A continuación detallamos los principales puntos del diseño (ver Figura 2.1):

- **S3 (Simple Storage Service)**

Es el servicio de almacenamiento masivo, transparente y de alta disponibilidad de Amazon. Aquí almacenamos nuestro dataset que son los Logs (input) y los resultados del proceso MapReduce (output).

- **EC2 (Elastic Compute Cloud)**

Este servicio de Amazon provee los recursos computacionales necesarios para correr nuestra aplicación MapReduce. En EC2 se implementa la computación paralela dado por Hadoop, donde los Map y Reduce se ejecutarán.

- **Host**

Es la máquina donde reside la aplicación gráfica donde se mostrará los gráficos.

- **Request**

Es el pedido que realiza el host a través del Internet a los servicios de Amazon. Cuando se requiere los datos para un gráfico en particular se genera un request.

- **Response**

Es la respuesta que el servicio de Amazon le provee al host a través del internet. En los request vienen los datos necesarios para un gráfico en particular.

Cuando la aplicación en el host reciba el response con los datos requeridos para realizar la gráfica, mediante la librería JFreeChart interpretamos los datos a gráficas.

- **Elastic MapReduce**

Es un servicio que ofrece Amazon en el que se puede programar la ejecución de tareas en EC2 y S3. Este servicio nos ayuda en la ejecución automática de los procesos MapReduce y la transferencia de archivos desde el S3 al EC2 y viceversa.

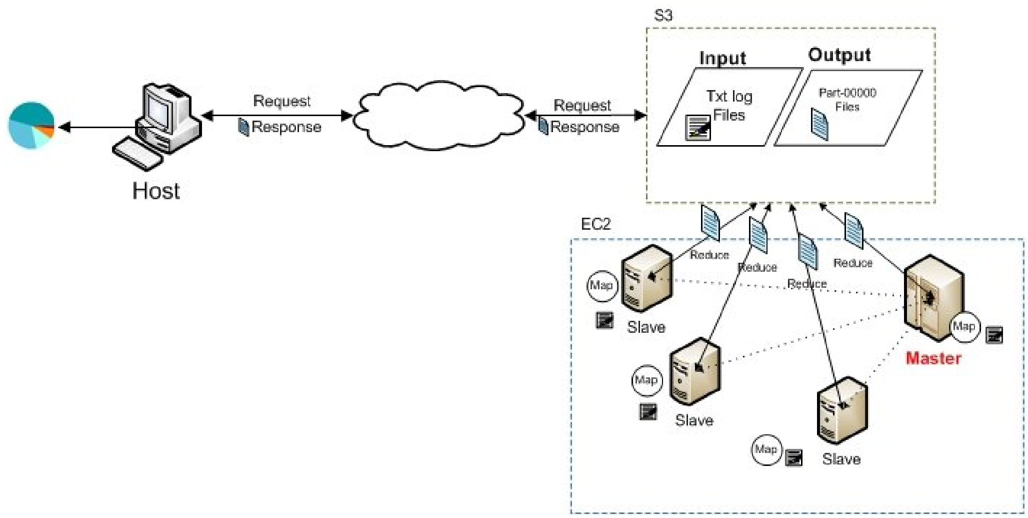


Figura 2.1 Diseño de la solución en Amazon Web Services

2.6. Diseño MapReduce implementando en Hadoop

Nuestro diseño se basa en los procesos principales del paradigma como son el proceso de Map y el proceso de Reduce. A continuación se describe cada uno de ellos.

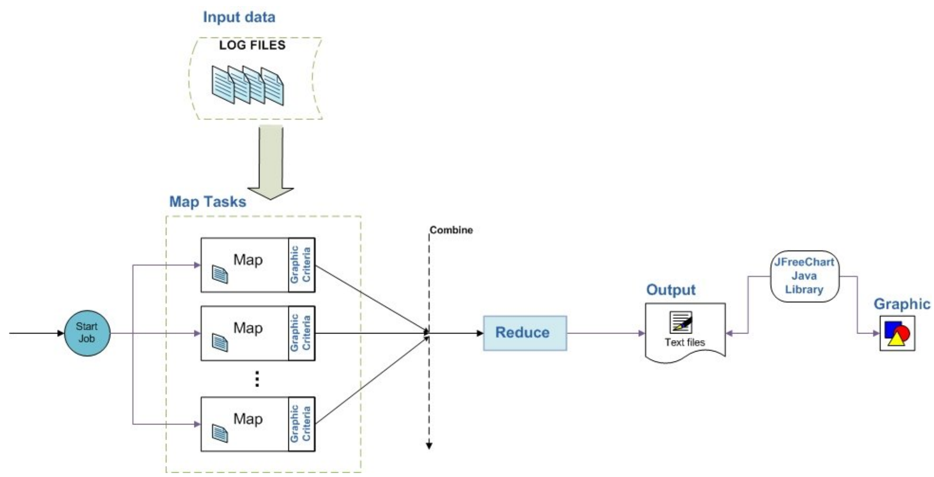


Figura 2.2 Diseño de la solución basada en MapReduce

Antes de aplicar el paradigma MapReduce realizamos un proceso de conversión de los archivos pcap hacia archivos de texto plano. Esta conversión pasó el contenido binario de los logs a texto, lo cual nos facilita el procesamiento de los datos en Hadoop.

2.6.1. Proceso Map

Una vez que se ejecuta el requerimiento de una gráfica se inicia el proceso de Map. Este proceso tiene lugar de la siguiente manera:

- Hadoop particiona el dataset en splits que se los tratará como texto. Creamos nuestra propia clase *MyPcap* que es el contenido de un paquete de nuestro log. Lo que recibe el Map son la el tipo de protocolo (TCP, IP, ICMP, ARP) y el valor es el objeto MyPcap que contiene los atributos más importantes de un paquete.

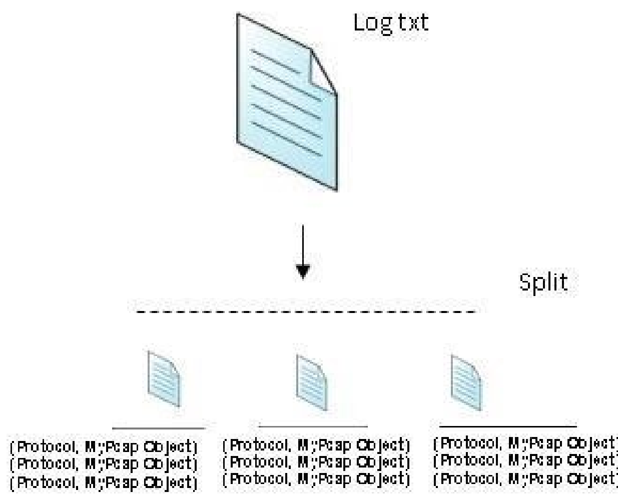


Figura 2.3 Split de un log en records (clave, valor)

- La función de Mapeo recibe los pares de (Protocolo, MyPcap Object), sobre estos valores se extrae los datos dependiendo de los criterios del gráfico a mostrar. Cuando el usuario ejecuta su requerimiento se envía un parámetro que guarda el número de gráfico deseado. Se recupera este parámetro en el Map para saber qué criterios debe aplicar y de esa manera obtener los datos que se requieren. Estos datos requeridos son enviados en forma de (graphic parameter, one), donde la clave es él o los valores que se consideran para necesarios para realizar la gráfica y one corresponde un número 1 para denotar la concurrencia de la misma.

Figura 2.4 Diseño del Map a la solución

2.6.2. Proceso Combine y Reduce

Los valores intermedios que produce el Map son ordenados por clave y una lista de valores que tienen en común a la clave. El proceso Reduce toma esta lista de valores con su clave y la cuenta teniendo la concurrencia total del mismo. Este proceso se realiza en los mismos nodos donde realiza el Map, y se le da el nombre de combine. El resultado del combine lo recibe el Reduce que realiza nuevamente la sumatoria y sus resultados son los finalmente los esperados.

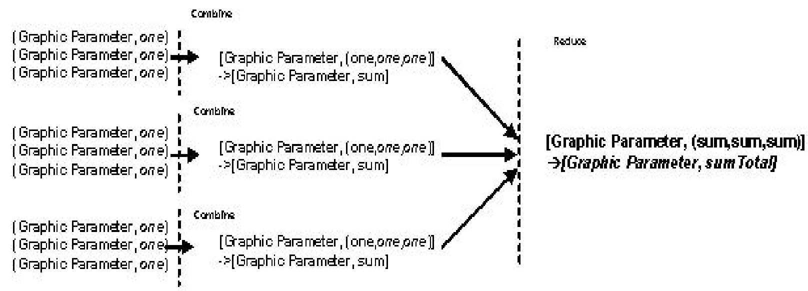


Figura 2.5 Diseño del Reduce de la solución

CAPÍTULO 3

3. IMPLEMENTACIÓN Y RESULTADOS

En el presente capítulo se explica con más detalle el desarrollo del sistema utilizando métodos y librerías para su implementación. Se presentan los tipos de gráficos realizados así como tiempos de ejecuciones utilizando Hadoop en los servicios de Amazon Web Services.

3.1. Conversión Logs pcap a archivos de texto plano

La conversión de Logs pcap a texto plano se realizó utilizando la librería JNetStream. Esta conversión no forma parte del proceso MapReduce sin embargo es necesario puesto que define el formato de los datos de entrada.

Se realizó un análisis exhaustivo al formato binario de los paquetes de los Logs pcap. Esto nos sirvió para acceder a los bits necesarios que contienen los datos relevantes para nuestro análisis.

```
if (linkType.equals("Ethernet") == true) {

    int etherProtocol = in.readUnsignedShort();

    // Now check if its IP protocol
    if (etherProtocol == 0x800) {
        int version = in.readBits(4);
        int hlen = in.readBits(4);
        int precedence = in.readBits(3);
        int delay = in.readBits(1);
        int throughtput = in.readBits(1);
        int reliability = in.readBits(1);
        in.readBits(2);     // Reserved 2 bits

        int length = in.readUnsignedShort();
        int id = in.readUnsignedShort();
        in.readBits(1);     // Reserved 1 flag bit

        int doNotFragment = in.readBits(1);
        int moreFragments = in.readBits(1);
```

Figura 3.1 Acceso a los bits del paquete con JNetStream

En la Figura 3.1 observamos como accedemos a los bits de los campos del paquete, dependiendo del tipo de protocolo accedemos a sus distintos campos. Imprimimos en un archivo de texto los campos que obtuvimos con el siguiente formato:

3.2. Esquema MapReduce

En esta sección se explica en detalle la implementación de MapReduce en Hadoop. Para que se ejecute MapReduce debe haber un pedido o request por parte del usuario.

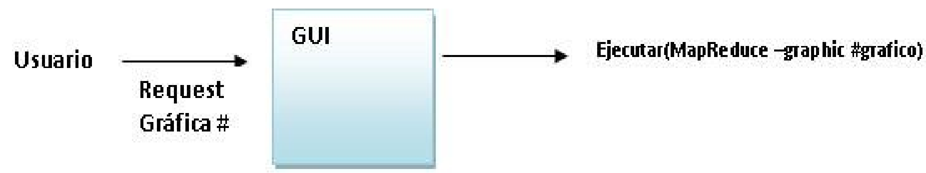


Figura 3.2 Ejecución MapReduce por parte del usuario

El usuario elige qué tipo de gráfica desea, el GUI ejecuta un Shell script donde le envía el parámetro: *-graphic #grafico_pedido.* Este parámetro es importante dentro del proceso MapReduce ya que denotará el tipo de criterios a considerar en la toma de datos específicos para realizar la gráfica.

3.2.1. Map

En este proceso realizamos la obtención de datos específicos para la gráfica. Obtenemos estos datos basándonos en el parámetro enviado –graphic *#grafico*. De esta forma nos aseguramos de obtener los datos correctos para la gráfica correcta.

Los datos de entrada al Map son (Protocolo, MyPcap Object), donde Protocolo es la clave y está en Texto, y MyPcap Object es el objeto que creamos para almacenar los datos de un paquete, podemos decir que el objeto MyPcap es un paquete personalizado con información relevante y de interés a nuestros propósitos. En la Figura 3.3 se muestra la clase MyPcap en java donde muestra los datos del paquete que guarda.

```
public class MyPcap implements
Writable{
        private int anio;
        private String mes;
        private int dia;
        private String hora;
        private String src_ip;
        private String dst_ip;
        private String src_port;
        private String dst_port;
        private String protocolo;
```

Figura 3.3 Clase MyPcap

Una vez que sabemos lo que recibe cada Map procedemos a obtener los datos. Para ello tenemos que saber los criterios que debemos considerar. Como hemos dicho los criterios depende de la solicitud del usuario, esta solicitud la sabemos obteniendo el valor del parámetro –graphic.

```
public void configure(JobConf job) {
                criterioBusqueda = Integer.parseInt(job.get("graphic"));
                }
```

Figura 3.4 Obtención del parámetro graphic

Sabiendo el parámetro enviado por el usuario podemos saber los criterios a usar.

Finalmente el Map envía los datos necesarios que cumple los criterios. Estos son enviados en la forma (clave, valor), donde la clave es el o los parámetros requeridos (ver Figura 3.5) y el valor es un entero *uno* bajo el tipo de dato IntWritable.

Criterio: Concurrencia de los días por cada mes

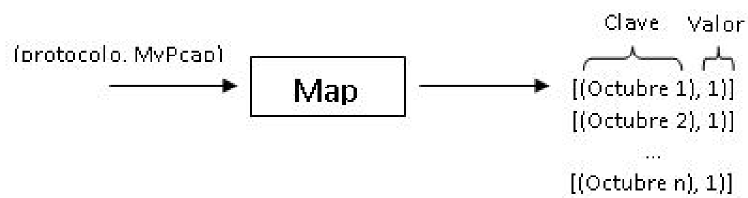


Figura 3.5 Implementación del Map de la solución

3.2.2. Reduce

La fase de Reduce se encarga del conteo de los valores que recibe por parte del Map. Este conteo lo hace a través de una sumatoria de unos que tienen una clave en común. Cuando este Reduce se lo hace en el mismo nodo en que se hace el Map toma el nombre de combine. En la Figura 3.7 se muestra la clave y valor que recibe el Reduce y lo que produce como resultado.

Siguiendo la secuencia de la Figura10

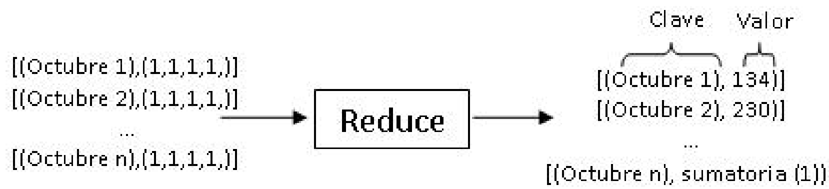


Figura 3.6 Implementación del Reduce de la solución

3.3. Creación de la GUI

Para el cumplimiento de la finalidad de nuestro proyecto como es el generar gráficamente reportes a partir de los Logs tcpdump, realizamos un GUI (Graphical User Interface) que facilite al usuario final la interacción con los resultados obtenidos en Hadoop.

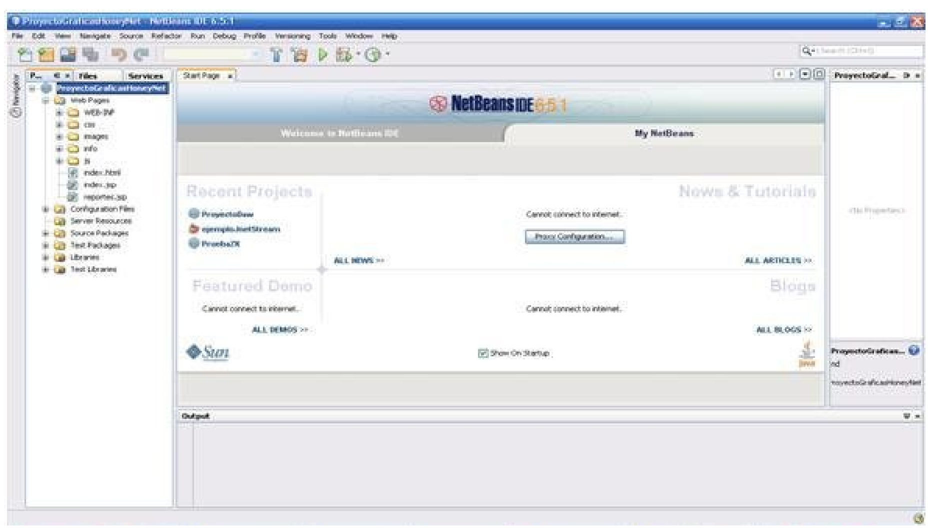


Figura 3.7 Netbeans IDE

En la Figura 3.7 observamos el IDE de Java *"Netbeans"* que fue el que utilizamos para la generación de nuestro GUI. La interfaz del proyecto fue elaborado vía web con la tecnología JSP (Java Server Pages) y Servlets. Debido a que las gráficas fueron elaboradas utilizando las librerías de Java *"JFreeChart"* surgió la necesidad de utilizar la tecnología JSP que utiliza código java en páginas web dándole dinamismo a la generación de gráficas.

3.3.1. Interfaz Desktop

La interfaz desktop se elaboró con el objetivo de que el usuario final no se preocupe en los comandos que debe utilizar para lograr una conexión y la ejecución de tareas de los servicios de Amazon. La interfaz se compone de dos secciones importantes como son la gráfica que se desea obtener y el número de nodos a utilizar. Cuando se ejecuta en la consola se registra un seguimiento del Job Flow, es decir se puede observar el estado cada cierto tiempo definido en este caso cada minuto. En la Figura 3.9 vemos el seguimiento completo de un Job Flow que se ejecutó en los servicios de Amazon.

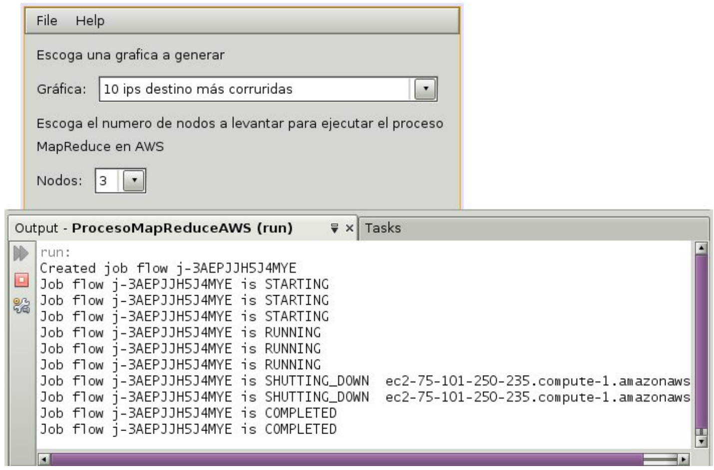


Figura 3.8 Interfaz Desktop en ejecución

3.3.2. Interfaz Web

Una vez realizado la ejecución en la interfaz desktop, los resultados generados ya han sido guardados en una carpeta destinada para los mismos y que se encuentran en el S3 de Amazon. La interfaz web realiza una extracción de estos resultados desde el S3 y con la utilización de la librería gráfica JFreeChart generamos la gráfica correspondiente.

Figura 3.9 Interfaz Web: Inicio

En la Figura 3.9 observamos la página principal del GUI, los colores utilizados hacen referencia de algún modo a la palabra honey (miel). El GUI está clasificado en 4 secciones:

- **Inicio:** Hace referencia al proyecto honeynet de manera general y el alcance de nuestro sistema (ver Figura 3.9).
- **Diseño:** Hace referencia al diseño MapReduce de nuestro sistema (ver Figura 3.10).

Figura 3.10 Diseño

- **Reportes:** En esta sección se muestra los tipos de gráficos a mostrar, para ello hemos colocado las opciones disponibles en radiobuttons para que se elija una opción a la vez. Junto a estas opciones se encuentra un botón “Generar” que toma la opción elegida y ejecuta los procesos necesarios como son: Ejecución de nuestro programa MapReduce en Amazon Web Services, llenado de resultados en una base datos, exposición de la gráfica mediante JFreeChart. La gráfica se muestra en una ventana diferente a la ventana principal (ver Figura 3.11).

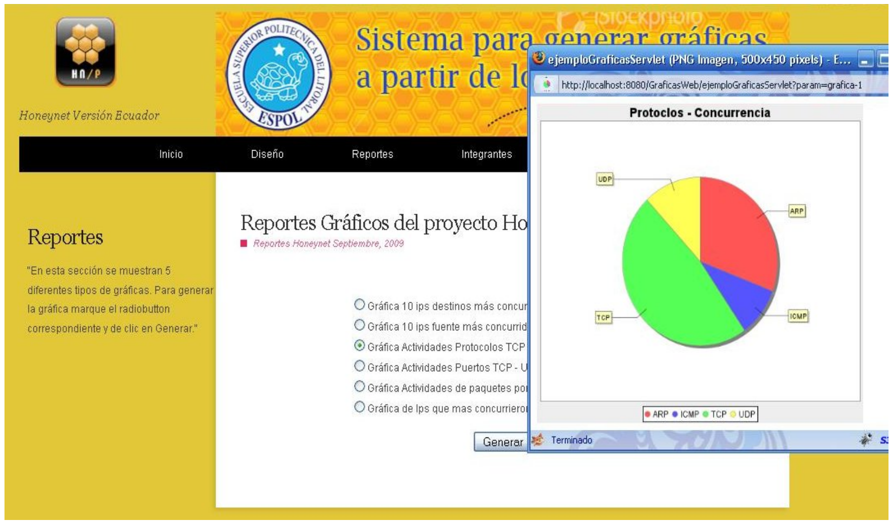


Figura 3.11 Reportes

- **Integrantes:** En esta sección se muestra a los estudiantes que participaron en la realización del sistema, se presenta sus nombres, números de matrícula y carrera (ver Figura 3.12).

Figura 3.12 Integrantes

3.4. Pruebas y Análisis de Resultados

Para las pruebas que se realizaron en el sistema, se tomó en consideración dos factores importantes como son: el número de nodos que conforman el clúster y el tamaño del dataset.

Debido a que el sistema trabajó sobre Elastic MapReduce, los tiempos que hemos obtenido son la sumatoria de los tiempos de cada uno de los pasos o “steps” que Elastic MapReduce realiza. Para ello hemos separado los tiempos en sus respectivos pasos.

	Tiempo			
No. Nodos	**Step 1 "S3 to HDFS"**	**Step 2 "MapReduce"**	**Step 3 "HDFS to S3"**	**Total**
5	66 minutos	62 segundos	21 segundos	67 min 23 seg
8	66 minutos	41 segundos	21 segundos	67 min 2 seg
10	65 minutos	35 segundos	23 segundos	65 min 58 seg

Tabla 1. Tiempos obtenidos sobre un dataset de 1.4GB

Tomando en cuenta el tiempo total de la Tabla 1 vemos que no hay mucha diferencia entre el número de nodos. Esto se debe a que estamos viendo el tiempo global de los 3 pasos que utiliza Elastic MapReduce. En el paso 1 se realiza la copia del S3 al HDFS, lo cual toma el mayor tiempo de los pasos debido a que el tamaño del dataset es relativamente grande. El paso 2 realiza la ejecución del proceso MapReduce sobre la data que ha sido almacenada en el HDFS. En el paso 3 se realiza la copia del archivo de resultados “part-00000” del HDFS al S3 lo cual por ser un archivo relativamente pequeño no toma mucho tiempo.

Enfocándonos en el paso 2 que es el más importante para analizar nuestro resultado, sacamos el siguiente gráfico.

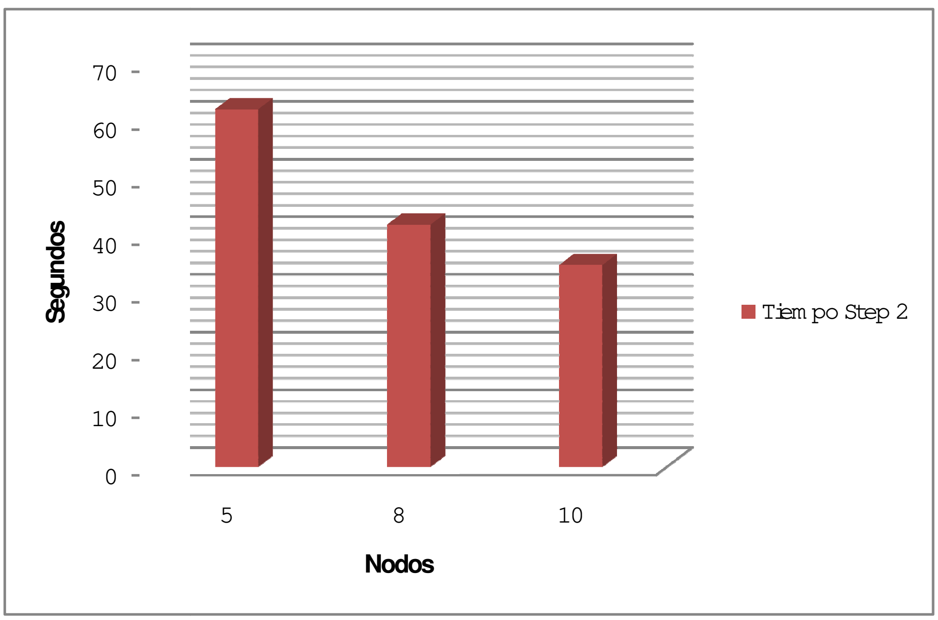


Figura 3.13 Tiempo de ejecución MapReduce sobre 1.4GB de data

Observando la figura 3.13 podemos notar la diferencia que existe en la utilización de varios nodos. Cuando usamos 5 nodos el tiempo fue de 62 segundos al duplicar el número de nodos el tiempo fue de 35 segundos, esta diferencia se debe a una de las características importantes del esquema MapReduce como es la programación paralela. La programación paralela hace que más datos se estén procesando al mismo tiempo puesto que existe mayor número de nodos, de ahí a que el tiempo de procesamiento tienda a disminuir conforme a la inclusión de más nodos.

3.4.1. Comparación frente a otras herramientas del mercado

Para realizar una comparación con otra herramienta que realice análisis de tráfico de red hemos escogido a Wireshark. La comparación se realizó bajo el concepto de la capacidad de procesamiento de paquetes sobre un tamaño de datos considerable.

Realizamos una prueba sobre una PC que tiene 3 GB de RAM, en la cual procesamos diferentes tamaños de dataset y obtuvimos los siguientes resultados:

	90MB	265MB	1GB
WIRESHARK	2 min	8min 18 seg	No fue posible
MAPREDUCE	47 seg	1 min 28 seg	6 min 3seg

Tabla 2. Tiempo de procesamiento de datos Wireshark vs MapReduce

Como podemos observar en la Tabla 2 el tiempo de procesamiento hecho en Hadoop es mucho menor al tiempo que realizó Wireshark. Además a medida que la data comienza a incrementar de tamaño, los tiempos en Wireshark van siendo significativamente mayores esto se debe a que para archivos mayores a 100 MB Wireshark tiende a ponerse lento en el procesamiento [9]. Además de estar limitada a la memoria RAM de la PC en la que se está ejecutando. Con MapReduce se reduce el tiempo de procesamiento de datos, en la Tabla 2 vemos que para 1GB de datos se tuvo un tiempo corto con respecto a Wireshark que no pudo terminar el procesamiento por la falta de memoria de la PC. Las ventajas de MapReduce sobre la herramienta Wireshark son que ésta última no posee un esquema de paralelismo que si lo tiene Hadoop. En situaciones en el que los datos a procesar son pequeños (hasta los 100 MB) el uso de Wireshark es justificado, no así cuando los datos a procesar son mucho más grandes (mayores a 100 MB) en ese caso una buena alternativa es la programación paralela implementada por Hadoop.

CONCLUSIONES Y RECOMENDACIONES

Conclusiones

1. El inconveniente que presentan las herramientas para el análisis de tráfico de red, es la capacidad de procesar grandes cantidades de datos. Esto limita a que no se realice un análisis completo sobre estos datos. El esquema MapReduce es una alternativa a este inconveniente, ya que implementa programación paralela en la que puede ejecutar grandes cantidades de datos sobre un grupo de computadoras o clúster.

2. El sistema para generar gráficas a partir de logs tcpdump implementa el esquema MapReduce a través del Framework de Apache Hadoop. Este sistema realiza un análisis en todo el dataset, obteniendo información de mayor alcance que si lo hiciéramos con una herramienta común para este tipo de análisis. El sistema ha sido desarrollado como una aplicación web con una interfaz sencilla en la que el usuario final puede interactuar sin ningún problema. Las pruebas del sistema se realizaron utilizando los servicios web de Amazon, lo cual nos ayudó mucho ya que pudimos probar el sistema con varios nodos es decir en un clúster real.

3. El servicio Elastic MapReduce que provee Amazon nos ayudó en la automatización de tareas como crear instancias, ejecución del proceso

MapReduce, copiar el dataset de S3 a EC2 y viceversa. De esta forma el usuario final no tiene que preocuparse en los comandos necesarios para utilizar los servicios de Amazon. La utilización de la librería gráfica JFreeChart nos proporcionó un buen manejo de los datos a mostrar, además de ofrecernos diferentes tipos de gráficos de acuerdo al tipo de análisis realizado.

4. Cabe recalcar que esta experiencia nos ayudó a fortalecer más los conocimientos adquiridos en la realización de aplicaciones web, así como la aplicación de nuevos conocimientos como el esquema MapReduce, los servicios de Amazon que sin ninguna duda nos servirán en el ámbito profesional.

Recomendaciones

Algunas de las grandes empresas como Google, Yazoo, IBM, entre otras, han apostado a la programación paralela como plataforma para muchos de sus servicios, esto nos da a entender que cada vez más va tomando importancia la implementación de servicios en este tipo de esquema. La utilización de nuevas alternativas de desarrollo resulta asequible por su bajo costo económico y su eficiencia en temas como el procesamiento masivo de datos.

1. El análisis que realizamos sobre la data es muy limitado y poco profundo, pero se puede realizar un análisis más exhaustivo en los datos como por ejemplo seguir un comportamiento de una dirección ip específica, para ello la inclusión de nuevas herramientas como datamining junto con el esquema Map reduce sería una alternativa muy buena [10].

BIBLIOGRAFÍA

[1] Pazmiño, M. y Avilés, J. “Captura y Análisis de los ataques informáticos que sufren las redes de datos de la ESPOL, implantando una honeynet con miras a mejorar la seguridad informática en redes de datos del Ecuador”. Tesis de grado en la ESPOL . Guayaquil, Marzo 2009.

[2] Acevedo, V. “Definición Dataset”,< http://www.gvsig.org/web/docdev/trunk-extensions-guide/otras-librerias/libreria-de-raster/terminologia > [Consulta: miércoles,15 de julio de 2009].

[3] Wireshark Foundation. “Wireshark”, < http://www.wireshark.org > [Consulta: martes 21 de julio 2009].

[4] Bernardczyk M. “JNetStream”, < http://jnetstream.com > [Consulta: sábado, 18 de julio de 2009].

[5] Dean, J. y Ghemawat, S. “MapReduce: Simplified Data Processing on Large Clusters“. En Memorias del Sixth Symposium on Operating System Design and Implementation (OSDI 2004). San Francisco, CA-EE.UU., Diciembre, 2004.

[6] The Apache Software Foundation. “Apache Hadoop”, < http://hadoop.apache.org > [Consulta: lunes, 20 de julio de 2009].

[7] Object Refinery Limited. “JFreeChart”, < http://www.jfree.org/jfreechart > [Consulta: lunes,20 de julio de 2009].

[8] Universidad Autónoma de México. “Bitácora Honeynet UNAM”, <http://www.honeynet.unam.mx/es/reports.pl> [Consulta: lunes 17 de agosto 2009].

[9] Wireshark Foundation “Wireshark Perfomance”, < http://wiki.wireshark.org/Performance > [Consulta: lunes 10 de septiembre 2009].

[10] Nauta, K., Lieble, F. “Offline Network Intrusion Detection: Mining TCPDUMP Data to Identify Suspicious Activity”. Presentado en AFCEA Federal Database Colloquium, San Diego, CA-EE.UU., Septiembre, 1999.

[11] Weigle, E., Feng, W. “TICKETing High-Speed Traffic with Commodity Hardware and Software”. En Memorias del Third Annual Passive and Active Measurement Workshop (PAM2002). Fort Collins, CO-EE.UU., Marzo, 2002.

[12] Barse, E. L., Jonsson, E. “Extracting attack manifestations to determine log data requirements for intrusion detection”. En Memorias del 20th Annual Computer Security Applications Conference. Diciembre, 2004.

[13] Abad, C., Taylor, J., Sengul, C., Yurcik, W., Zhou, Y., Rowe, K. “Log correlation for intrusion detection: a proof of concept”. En memorias del 19th Annual Computer Security Applications Conference. Diciembre, 2003.

[14] Smith, FD., Campos, FH., Jeffay, K., Ott, D. “What TCP/IP protocol headers can tell us about the web”. Presentado en ACM SIGMETRICS Performance Evaluation Review, New York, NY-EE.UU.,Junio,2001.

[15] Thakar, U., Varma, S., Ramani, AK. “Honey Analyzer – Analysis and Extraction of Intrusion Detection Patterns & Signatures Using Honeypot”. Presentado en The Second International Conference on Innovations in Information Technology, 2005.

ANEXOS

ANEXO A: Código Java del proceso de mapeo en Hadoop

```
public class sg2map extends MapReduceBase
implements Mapper<Text , MyPcap, Text,  IntWritable> {

private final static IntWritable one = new IntWritable(1);
private final static Text gparameter = new Text();
private final static Text error = new Text("ERROR");
private int numBusqueda = 0;



public sg2map() { }


public void configure(JobConf job) {
             numBusqueda = Integer.parseInt(job.get("graphic"));
             }

public void map(Text key, MyPcap value, OutputCollector<Text,
IntWritable> output,
  Reporter reporter) throws IOException {

 realizarBusqueda(key.toString(),numBusqueda,value,output,reporter);

}


public static void realizarBusqueda(String key,int
numeroBusqueda,MyPcap pcap,OutputCollector<Text, IntWritable>
output, Reporter reporter) throws IOException{

  StringBuilder toReturn = new StringBuilder();

  switch(numeroBusqueda){

  case 1:   if(key.compareTo("null")!=0){
                  gparameter.set(pcap.getDst_ip());
                  output.collect(gparameter,one);
                  }
                  break;

  case 2:   if(key.compareTo("null")!=0){
                  gparameter.set(pcap.getSrc_ip());
                  output.collect(gparameter,one);
                  }
                  break;
```

```
  case 3:   if(key.compareTo("null")!=0 &&
key.compareToIgnoreCase("udp")!=0){
                  gparameter.set(key);
                  output.collect(gparameter,one);
                  }
                  break;


  case 4:   if((key.compareToIgnoreCase("tcp")==0
                        || key.compareToIgnoreCase("udp")==0) &&
key.compareTo("null")!=0){

      if(pcap.getDst_port().trim().compareTo("445")==0 ||
               pcap.getDst_port().trim().compareTo("6667")==0 ||
               pcap.getDst_port().trim().compareTo("80")==0 ||
               pcap.getDst_port().trim().compareTo("21")==0 ||
               pcap.getDst_port().trim().compareTo("23")==0 ||
               pcap.getDst_port().trim().compareTo("22")==0 ||
               pcap.getDst_port().trim().compareTo("137")==0 ||
               pcap.getDst_port().trim().compareTo("138")==0 ||
               pcap.getDst_port().trim().compareTo("1434")==0) {
                        toReturn.append(pcap.getProtocolo());
                        toReturn.append("\t");
                        toReturn.append(pcap.getDst_port());
                        gparameter.set(toReturn.toString());
                        output.collect(gparameter,one);}
                        }
                        break;

  case 5:   if (key.compareTo("null") != 0) {
                  if (pcap.getMes().compareToIgnoreCase("agosto") ==
0) {
                        toReturn.append(pcap.getMes());
                        toReturn.append("\t");
                        toReturn.append(pcap.getDia());
                        gparameter.set(toReturn.toString());
                        output.collect(gparameter, one);}
                        }
                        break;

  case 6:   if (key.compareTo("null") != 0) {
                  if
(pcap.getMes().compareToIgnoreCase("septiembre") == 0) {
                        toReturn.append(pcap.getMes());
                        toReturn.append("\t");
                        toReturn.append(pcap.getDia());
                        gparameter.set(toReturn.toString());
                        output.collect(gparameter, one);}
                        }
                        break;


  case 7:   if (key.compareTo("null") != 0) {
```

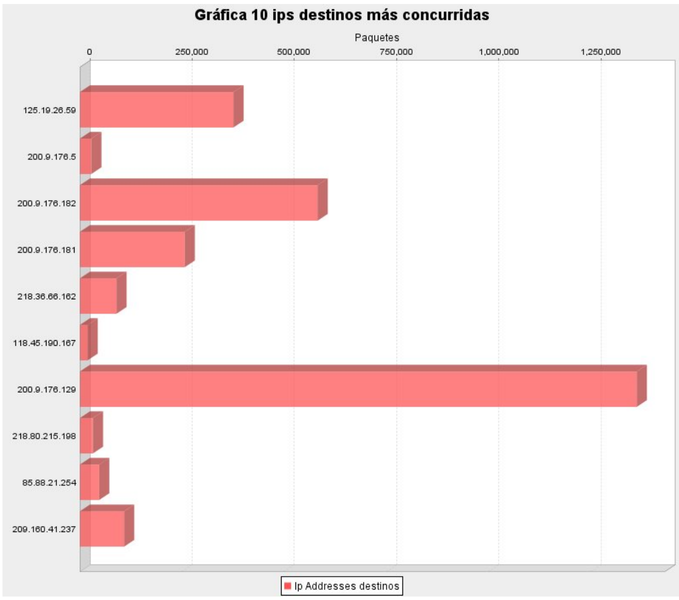
```
                  if (pcap.getMes().compareToIgnoreCase("octubre")
== 0) {
                        toReturn.append(pcap.getMes());
                        toReturn.append("\t");
                        toReturn.append(pcap.getDia());
                        gparameter.set(toReturn.toString());
                        output.collect(gparameter, one);}
                        }
                        break;

  case 8:   if (key.compareTo("null") != 0) {
                  if (pcap.getMes().compareToIgnoreCase("noviembre")
== 0) {
                        toReturn.append(pcap.getMes());
                        toReturn.append("\t");
                        toReturn.append(pcap.getDia());
                        gparameter.set(toReturn.toString());
                        output.collect(gparameter, one);}
                        }
                        break;

  default:        output.collect(error,one);
                  break;
  }
}
```

ANEXO B: Código Java del proceso de reduce en Hadoop

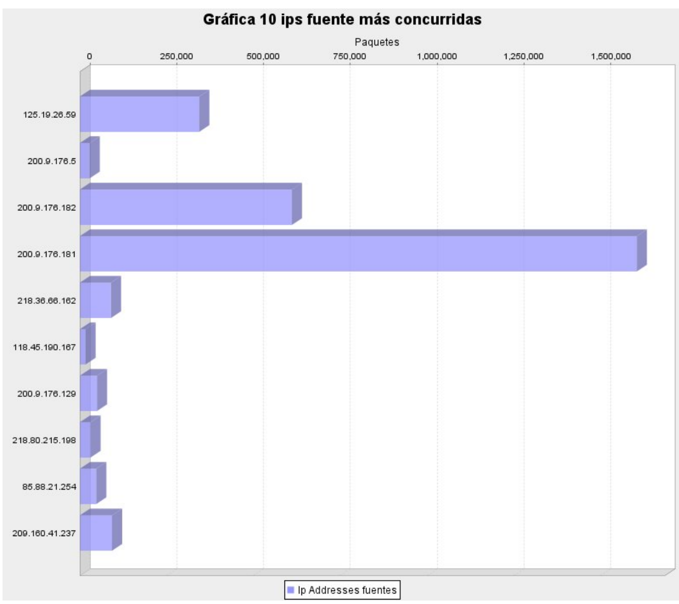
```
public class sg2reduce extends MapReduceBase
implements Reducer<Text, IntWritable, Text, IntWritable> {

public sg2reduce() { }

public void reduce(Text key, Iterator<IntWritable> values,
  OutputCollector<Text, IntWritable> output, Reporter reporter)
      throws IOException {

  int sum = 0;
  while (values.hasNext()) {
            sum += values.next().get();
      }
      output.collect(key, new IntWritable(sum));

      }
}
```

ANEXO C: Gráficas Generadas

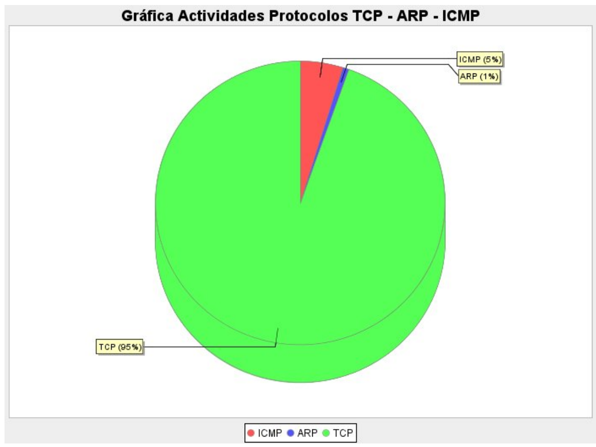
Gráfica 10 ips destinos más concurridas



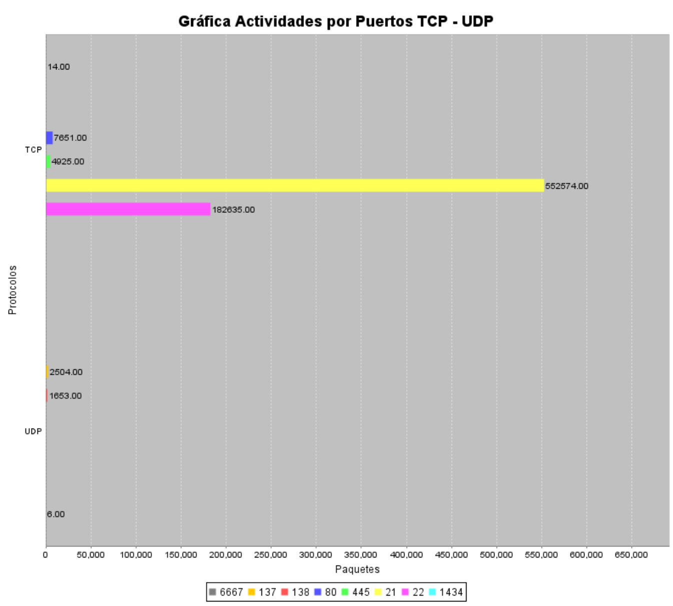
Gráfica 10 ips fuente más concurridas



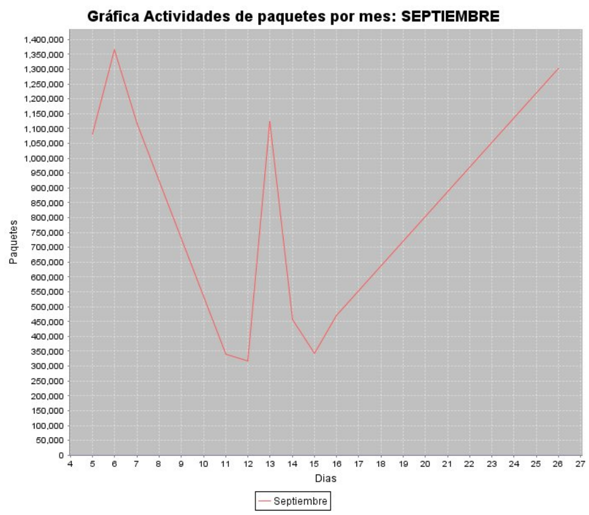
Gráfica Actividades Protocolos TCP - ARP – ICMP



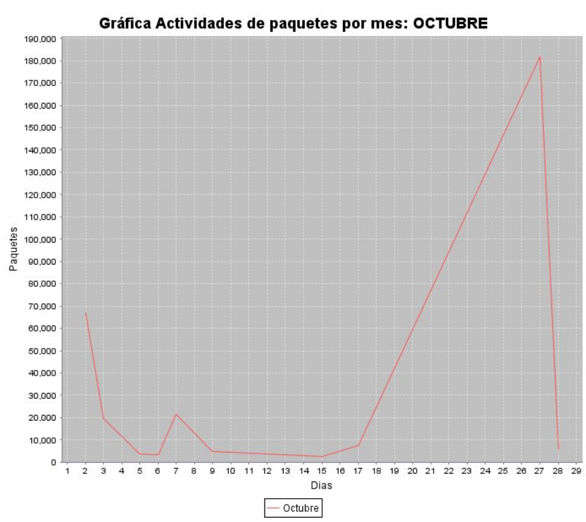
Gráfica Actividades por Puertos TCP – UDP



Gráfica Actividades de paquetes por mes: SEPTIEMBRE



Gráfica Actividades de paquetes por mes: OCTUBRE



Gáfica Actividades de paquetes por mes: NOVIEMBRE